Space-efficient algorithm for computing a centerpoint of a point Set in $\mathbb{R}^2$

Binay K. Bhattacharya* Subhas C. Nandy† Sasanka Roy‡

Abstract: We study space-efficient algorithm for computing a centerpoint for a set P of n points in $\mathbb{R}^2$, where the points in P are given in a read-only array. We propose an algorithm that finds a centerpoint of P in $O(T(n^2)\log^2 n)$ time using $O(S(n^2))$ extra space, where $T(n)$ and $S(n)$ are the time and extra space complexities of computing the median among a set of n elements stored in a read-only array.

Keywords: centerpoint, space-efficient algorithm, prune-and-search, duality, arrangement.

1 Introduction

The *k-center* of a set P of n points in $\mathbb{R}^d$ is the set C of points in $\mathbb{R}^d$ such that for any half-plane that passes through a point $c \in C$ contains at least k points. The set C is said to be k-hull of the points in P, and any point $c \in C$ is called a *k-centerpoint* of P. In $\mathbb{R}^d$, if $k = \lceil \frac{n}{d+1} \rceil$, then a point in C is referred to as *centerpoint* of P. It can be shown that a centerpoint of a point set always exists [8].

In $\mathbb{R}^2$, Cole et al. [6] proposed an algorithm that runs in $O(n\log^5 n)$ time. The algorithm exploits its connection with the k-hull of the point set P. Matousek [14] improved the time complexity to $O(n\log^4 n)$. Cole [5] further improved the time complexity to $O(n\log^3 n)$ using the parametric searching technique [15]. Naor and Sharir [19] extended this to $\mathbb{R}^3$ by proposing an algorithm that runs in $O(n^2\log^6 n)$ time. Finally, Jadhav et al. [11] proposed an optimal linear time algorithm for the problem in $\mathbb{R}^2$ using prune-and-search technique.

A relevant variation, where an approximate centerpoint is desired, has also been studied in the literature. Megiddo [16] showed that given the point set P in $\mathbb{R}^2$, one can compute a point in $O(n)$ time so that both the halfplanes defined by any line passing through that point contain at least $\frac{n}{4}$ points. Matousek [13] showed that one can compute an approximate center in $\mathbb{R}^2$ in linear time, which is as close to the optimum as desired.

In this paper, we assume that the planar point set P is given in read-only array. Using geometric duality [8], we propose an $O(T(n^2)\log^2 n)$ time algorithm for computing a k-centerpoint (if exists) in this environment, which uses $O(S(n^2))$ extra space, where $T(n)$ and $S(n)$ are the time and extra space complexities for computing the median among a set of n elements stored in a read-only array. Since the centerpoint (i.e., k-centerpoint for $k = \lceil \frac{n}{3} \rceil$) always exists [8], the asymptotic complexity results for computing centerpoint are $O(T(n^2)\log^2 n)$ time and $O(S(n^2))$ extra space.

2 Preliminaries

We will use P' and $\mathcal{A}(P')$ to denote the dual lines corresponding to the members in P and the arrangement of the lines in P', respectively. Let L_k denote the k-th level of the arrangement, and λ_k denote the convex hull of L_k. Note that, L_k is a x-monotone polychain for any $k = 1, 2, \ldots, n$. We exhaustively enumerate the points of intersection of all pairs of members in P' to find the leftmost and rightmost vertices v_L and v_R of $\mathcal{A}(P')$. Let V_L and V_R be the vertical lines through v_L and v_R respectively. An easy way to compute the k-centerpoint in $\mathbb{R}^2$ is as follows.

> Find a line ℓ (if exists) in the dual plane that does neither intersects L_k nor $L_{(n-k)}$. The dual point π of the line ℓ in the primal plane is a k-centerpoint of P.

Since the combinatorial complexity of the arrangement of P' is $O(n^2)$ [8] and that of the k-th level of the arrangement of P' is $O(nk^{1/3})$ [7], the line ℓ can easily be computed in $O(n^2)$ time using $O(nk^{1/3})$ extra space. Our objective is to devise an algorithm for finding such a dual line ℓ that uses very small amount of extra space.

During the discussion of this algorithm, we will exhaustively use the algorithm for computing the k-th order statistic of a set of n numbers given in a read-only array. Table 1 gives an exhaustive set of results available in the literature [4]. We will use $T(n)$ and $S(n)$ to denote the time and extra space complexities of computing k-th order statistics for a set of n real numbers stored in a read-only array.

We now describe a method of generating the polychain L_k in space-efficient manner, which will be used in our proposed algorithm.

*School of Computing Sciences, Simon Fraser University, Canada, binay@sfu.ca

†Indian Statistical Institute, Kolkata, India, nandysc@isical.ac.in

‡Chennai Mathematical Institute, Chennai, India, sasanka@cmi.ac.in

Table 1: Complexity results for median finding algorithms when input is in a read-only array

Time ($T(n)$)	Space ($S(n)$)	Reference
$O(n^{1+\epsilon})$	$O(\frac{1}{\epsilon})$, where $2\sqrt{\frac{\log\log n}{\log n}} \leq \epsilon < 1$	[18]
$O(n \log^2 n)$	$O(\log n)$	[20]
$O(n \log_s n + n \log s)$	s, where $s \geq \log^2 n$	[10, 17]
$O(n)$	s, where $s \geq n/\log n$	[9]
$O(sn^{1+1/s} \log n)$	s, where $s \leq \log n$	[20]
$O(n \log\log_s n)$ (randomized)	s (any value)	[2, 18]
$O(n \log_s U)$	s (any value)	[3]
$O(n \log n \lceil \log_s \log U \rceil)$	s (any value)	[3]

2.1 Generation of L_k

Let v_0^k be the k-th point of intersection of the members of P' on the line V_L from below. This is the leftmost vertex of L_k and hence of λ_k. Also, we know the line $\ell \in P'$ that contains the edge $e \in L_k$ incident on v_0^k. Again, we spend $O(n)$ time to compute the intersections of the lines in $P' \setminus \{\ell\}$ with the line ℓ to identify the intersection point (say v_1^k) closest to v_0^k. Thus, we can identify the vertices and edges of L_k in linear order from left to right. The point v_0^k can be identified in $O(nk)$ time using $O(1)$ extra space[1] by inspecting the points of intersection of the members of P' with the line V_L where the lines in P' are available in their corresponding dual form (as the points in the primal plane) in the read-only array P. Since identifying each vertex of L_k needs $O(n)$ time, and the number of vertices in L_k can be $O(nk^{1/3})$ in the worst case [7], the generation of L_k needs $O(n^2k^{1/3})$ time in the worst case using $O(1)$ extra work-space.

2.2 Useful results

In order to describe our algorithm, the following results are important.

Lemma 1 *Given a collection P' of n lines in $\mathbb{R}^2$ in a read-only array, and a query line ℓ, one can find all the vertices of L_k lying on ℓ in $O(n^2)$ time using $O(1)$ extra work-space.*

Proof. We can process the intersection points of the lines of P' with ℓ in increasing order. Each intersection point π can be found $O(n)$ time, and the level of the point π in the arrangement $\mathcal{A}(P')$ can be computed in another $O(n)$ time. Since there are n number of intersection points of the members of P' with ℓ, the result follows. □

Lemma 2 *Given a collection P' of n lines in $\mathbb{R}^2$ in a read-only array, and a query line ℓ, it is possible to determine in $O(n^2)$ time using $O(1)$ work-space whether*

(i) ℓ touches λ_k, or

(ii) ℓ intersects the interior of λ_k, or

(iii) ℓ does not intersect λ_k at all.

Proof. Using Lemma 1, one can compute the level of each intersection point of ℓ with the members in P'.

- If the level of all the intersection points is strictly greater than k, then ℓ does not intersect λ_k at all.
- If there exists at least one intersection point whose level is less than k, then ℓ intersects the interior of λ_k.
- Now, we need to consider the case where the level of all the intersection points is greater than or equal to k. Since L_k is a x-monotone polychain, the number of intersection points of ℓ and L_k is even, and these intersection points appear in (disjoint) pair while observing the intersection points of the members of P' and ℓ in order. If every (disjoint) pair[2] of intersection points are coincident (i.e., a vertex of L_k) then ℓ touches λ_k, otherwise ℓ intersects λ_k.

□

3 Algorithm

Lemma 3 *Given a slope m, one can compute a vertex z of λ_k, such that the line $\ell(z, m)$ of slope m and passing through z that touches λ_k, in $O(n^2k^{1/3})$ time using $O(1)$ work-space.*

Proof. The vertex z can be identified by (i) generating the polychain L_k, (ii) observing the y-intercept[3] of the line $\ell(v, m)$ for each vertex $v \in L_k$, and (iii) reporting the vertex z such that the y-intercept of the line $\ell(z, m)$ is maximum. In Subsection 2.1, we have shown that L_k can be generated in $O(n^2k^{1/3})$ time using $O(1)$ space. Thus, the result follows. □

[1]Note: The point v_0^k can also be identified in $O(T(n))$ time using $O(S(n))$ extra space by invoking any median finding algorithm with input in readonly array. It does not change the time complexity of generating L_k; rather increases the extra space complexity. However, in the entire algorithm, the time and extra space complexity remains same for using any one of these algorithms.

[2]consecutive odd and even numbered intersection points

[3]By y-intercept of a line ℓ, we mean the distance of the point of intersection of the line ℓ and the y-axis from the origin.

Lemma 4 *Given a slope m, one can compute a vertex z of λ_k, such that the line $\ell(z, m)$ of slope m and passing through z that touches λ_k, in $O(T(n^2)\log n)$ time using $S(n^2)$ work-space.*

Proof. Consider the lines with slope m at each vertex of the arrangement $\mathcal{A}(P')$. Let B be the set of y-intercepts of these lines arranged in order. We perform binary search among the elements of B to identify a line that is a tangent of λ_k.

In order to justify the time complexity, observe that, here the elements of B can be generated online; each element corresponds to a vertex of $\mathcal{A}(P')$, or in other words, it corresponds to a tuple (i, j) where the lines $P[i]$ and $P[j]$ generate that vertex. Thus, these elements can be assumed to be available in a hypothetical read-only array B of size $O(n^2)$.

Also observe that, the elements of B are not ordered with respect to the desired search key. Thus, at each step, (i) we identify the r-th smallest element α in the array B for some desired integer r by executing an appropriate algorithm for computing the order statistics (see Table 1) on the points of intersection of the lines in P' with the y-axis, and then (ii) testing whether the line $y = mx + \alpha$ *properly intersects* or *touches* or *above* λ_k using Lemma 2. Since $|B| = O(n^2)$ in the worst case, each step of binary search takes $O(T(n^2)+n^2)$ time. As $T(n)$ is slightly super linear, the result follows. □

A simple method of computing the k-centerpoint is as follows:

Step 1: Traverse L_k in order to count its number of vertices N_k, and set $a = 0$, $b = N_k$.

Step 2: Again traverse L_k in order to choose the $\lceil \frac{a+b}{2} \rceil$-th vertex of L_k.

Step 3: Identify the edge $e = (\alpha, \beta)$ of λ_k that intersects the vertical line $V_{\lceil \frac{a+b}{2} \rceil}$ using the stack-based algorithm of Barba et al. [1] for computing convex hull of an x-monotone polychain in read-only environment. This is a tangent of λ_k.

Step 4: Let the line ℓ contain e. Now,

- **Step 4.1:** if the line ℓ does not properly intersect $\lambda_{(n-k)}$, then the line ℓ corresponds to a k-centerpoint of the point set P in the primal plane. Exit with the k-centerpoint.
- **Step 4.2:** if the edge e intersects $\lambda_{(n-k)}$, then k-centerpoint does not exist for the point set P. Exit with failure.
- **Step 4.3:** If ℓ intersects $\lambda_{(n-k)}$ to the left of α and also to the right of β, then also the k-centerpoint does not exist. Exit with failure.
- **Step 4.4:** if $\lambda_{(n-k)}$ intersects the line ℓ only to the left of α (resp. to the right of β), then any straight line touching λ_k and having slope less (resp. greater) than that of ℓ can not be a tangent of $\lambda_{(n-k)}$. We set $b = \frac{a+b}{2}$ (resp. $a = \frac{a+b}{2}$) to increase (resp. decrease) the slope of the line for a tangent (if possible).

Step 5: If $a < b$, then repeat Steps 2, 3 and 4.

Step 6: If $a = b$, then it implies that it is possible to draw tangent of $\lambda_{(n-k)}$ from the only vertex a of λ_k.

- We draw the vertical line V_a (from the vertex $a \in \lambda_k$). We compute the edge $e' = (\alpha', \beta') \in \lambda_{(n-k)}$ that is intersected by V_a (see Step 3).
- Let ℓ' be the line containing edge e'. We first test whether the line ℓ' intersects λ_k or not. If not, then the dual of ℓ' is a k-centerpoint. Otherwise, set either $(a' = -\infty, b' = \alpha')$ or $(a' = \beta', b' = \infty)$ depending on whether ℓ' intersect λ_k to the right or left extension of e'.
- Execute the same procedure on $\lambda_{(n-k)}$ to identify an edge of $\lambda_{(n-k)}$ (between the vertices a' and b') that corresponds to the k-centerpoint.

Step 7: If the line containing an edge of $\lambda_{(n-k)}$ is a tangent of λ_k, then we report it as k-centerpoint.

Step 8: Otherwise, if $a' = b'$ is attained, then a tangent of λ_k and $\lambda_{(n-k)}$ can only be drawn from the vertex $a' = b' \in \lambda_{(n-k)}$, and we test whether the line ℓ^*, containing a and a', is a tangent of both λ_k and $\lambda_{(n-k)}$. If so, then we report the dual of ℓ^* as the k-centerpoint; otherwise, we report the failure.

The time complexity of a single iteration is dominated by Step 3, which takes $O(nN_k \log n)$ time using $O(\log n)$ space [1], where N_k is the number of vertices in L_k. Since N_k can be $O(nk^{1/3})$ in the worst case [7], we have the following result:

Lemma 5 *Given a set P of n points in a read-only array, the k-centerpoint of the points in P can be computed in $O(n^2k^{1/3}\log^2 n)$ time using $O(\log n)$ extra space.*

We now describe a faster prune and search based method for computing a tangent line ℓ of λ_k that does not intersect $\lambda_{(n-k)}$.

3.1 Faster method

We can invoke an appropriate median finding algorithm (see Table 1) to compute a vertex $u \in \mathcal{A}(P')$ such that u_x (x-coordinate of the vertex u) is the median among

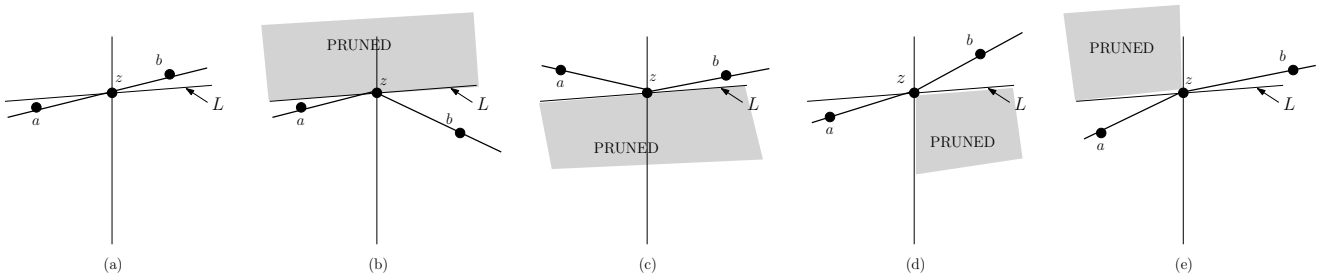

Figure 1: Pruning procedure for computing the ham-sandwich cut of two sets of linearly separable points

the x-coordinates of all the vertices in $\mathcal{A}(P')$. Now, we execute Algo-BRIDGE (described in Subsection 3.2) to identify the edge $e = (a, b) \in \lambda_k$ that intersects the vertical line V_u drawn at the point u. Let ℓ be the line containing the edge e. The different situations that can arise here are similar to those described in Steps 1-4 of the earlier algorithm. If the situation as in Step-4 arises, where $\lambda_{(n-k)}$ intersects ℓ to the right of b (resp. to the left of a), we repeat the same procedure with the vertices of $\mathcal{A}(P')$ to the right of b (resp. to the left of a). Since, in each level of recursion, we can discard a constant fraction of vertices in $\mathcal{A}(P')$, it takes at most $O(\log n)$ recursive calls to get the desired slope m.

Finally, we may arrive at a situation where the lines corresponding to both the edges incident at a vertex $u \in \lambda_k$ intersect $\lambda_{(n-k)}$, but these edges do not intersect $\lambda_{(n-k)}$ (see Step 6 of the earlier algorithm). As in the earlier procedure, here the tangent of $\lambda_{(n-k)}$ may be drawn only from the vertex u. We execute the same procedure on $\lambda_{(n-k)}$ to get the k-centerpoint. As mentioned in Step 8 of the earlier algorithm, here also we may arrive at a vertex $u' \in \lambda_{(n-k)}$ such that the tangent of λ_k may only exist from the vertex u'. Here, if the line ℓ^* containing u, u' is tangent of both λ_k and $\lambda_{(n-k)}$, we report ℓ^* as the k-centerpoint.

3.2 Algo-BRIDGE

In this section, we describe a procedure for computing an edge (called *bridge*) of λ_k that is intersected by a vertical line drawn at a vertex of $\mathcal{A}(P')$.

Lemma 6 *Given a point z on a vertical line V, one can compute the tangent of λ_k from the point z in one side of the vertical line V in $O(T(n^2)\log n)$ time using $S(n^2)$ work-space.*

Proof. Let us consider the left side of the line V. Let $\mathcal{U}$ be the set of vertices of $\mathcal{A}(P')$ to the left side of V. Let Π be the set of slopes of the lines $\ell_{(z,v)}$ (the line passing through the points z and v), for all $v \in \mathcal{U}$. We perform binary search among the members of Π to find a tangent of λ_k from the point z.

The time complexity of this procedure consists of two parts: (i) choosing a member $\pi = slope(\ell_{(z,v)})$ of Π, and (ii) checking whether the corresponding line is a tangent of λ_k.

In each step of the binary search, choosing $\pi \in \Pi$ takes $O(T(n^2))$ time with $O(S(n^2))$ space (see Table 1). The checking of whether the line drawn from z with angle π is a tangent of λ_k or above (resp. below) λ_k takes $O(n^2)$ time (see Lemma 1). After the checking in Step (ii), if it results a tangent, then the procedure stops. Otherwise, if it is above λ_k, then we consider $\pi' \in \Pi$, where $\pi' < \pi$; if it is below λ_k, then we consider $\pi' \in \Pi$, where $\pi' > \pi$.

We may have to repeat the procedure $O(\log n)$ times. Thus, the overall time complexity is $O(T(n^2)\log n)$ in the worst case. The space complexity follows from that of the median finding algorithm. □

Let the edge e of λ_k intersect the vertical line V at the point z^*. We apply the prune-and-search technique for computing z^*. Let $\mathcal{U}_{left}$ and $\mathcal{U}_{right}$ be the set of vertices of $\mathcal{A}(P')$ to the left and right of V; $\mathcal{U}_{left} \bigcup \mathcal{U}_{right} = \mathcal{U}$, where $|\mathcal{U}| = O(n^2)$, and $|\mathcal{U}_{left}| = |\mathcal{U}_{right}|$. Initially each vertex of $\mathcal{U}_{left} \bigcup \mathcal{U}_{right}$ is active.

We draw a line L that splits both $\mathcal{U}_{left}$ and $\mathcal{U}_{right}$ into two equal halves. Let it intersect V at a point z. We identify two vertices $a \in \mathcal{U}_{left}$ and $b \in \mathcal{U}_{right}$ such that the line $\ell_{(z,a)}$ and $\ell_{(z,b)}$ are tangents of λ_k (see the bold lines in Figure 1). If the three points z, a, b are collinear (see Figure 1(a)), then $z^* = z$, or in other words, the line containing z, a, b is the desired bridge. Otherwise, four situations can take place as shown in Figure 1(b-d) (ignoring the symmetric cases). In Figure 1(b) (resp. Figure 1(c)), z^* appears below (resp. above) z. Thus, the tangent from z^* can not touch the vertices of λ_k (if any) lying in the shaded portion. Thus, in the next iteration for finding z^*, we can prune the vertices of both $\mathcal{U}_{left}$ and $\mathcal{U}_{right}$ lying in the shaded portion. However, such a favorable event may not happen in the other two situations. In Figure 1(d) (resp. Figure 1(e)) z^* appears above (resp. below) z, and the tangent of λ_k from z^* to the left (resp. right) may be incident to a vertex in the 2nd (resp. 4th) quadrant defined by the lines L and V. Thus, only half of the vertices from $\mathcal{U}_{left}$ or $\mathcal{U}_{right}$ can be pruned.

In our prune and search algorithm, in each step we compute the line L using the algorithm of Megiddo [16] for computing the ham-sandwich cut of two point sets separated by a vertical line. This partitions the existing (not-pruned) points in both $\mathcal{U}_{left}$ and $\mathcal{U}_{right}$ in two equal parts. After computing the tangent, half of the points in at least one part must be pruned. The algorithm terminates either (i) the tangent line of λ_k from z in both the sides of V are same, or (ii) one of the parts ($\mathcal{U}_{left}$ or $\mathcal{U}_{right}$) contains a single point. In situation (ii), we compute the desired tangent from that single not-pruned point lying in one side of V to λ_k to the portion of λ_k in the other side of V.

Since in each iteration half of the existing vertices in $\mathcal{U}_{left}$ or $\mathcal{U}_{right}$ is surely pruned, in $O(\log n)$ iterations all but one vertices are pruned from $\mathcal{A}(P')$ those lie in one side of V. The number of intermediate iterations that has pruned vertices of the other half is also at most $O(\log n)$. Thus, the total number of iterations is $O(\log n)$ in the worst case.

3.2.1 Computing the ham-sandwich cut

Let R be a set of n red points and B be a set of n blue points lying in the two sides of a vertical line V. R and B are given in two read-only arrays. Our objective is to find a line L that splits the points in both R and B in two equal halves. We use the prune and search based algorithm of Megiddo [13] by tuning it to work in read-only environment.

We assume that the number of points in both R and B are odd. If not, we include one point in each of the sets. This ensures that L must pass through a point in both R and B respectively. Let R' (resp. B') be the dual lines of R (resp. B), and will be referred to as the *red-lines* (resp. *blue-lines*).

Assuming V as the y-axis, R and B have negative and positive x-coordinates respectively. Thus, the red lines have negative slopes and blue lines have positive slopes. Let LB and LR be the two monotone polychains, which represent the median level of the arrangement of $\mathcal{A}(R')$ and $\mathcal{A}(B')$ respectively. LR is a monotone decreasing function, and LB is a monotone increasing function. The ham-sandwich cut line L of the points in R and B correspond to the point of intersection of LB and LR.

Lemma 7 *Given the points R and B in two read-only arrays and a vertical line V, the side of V containing the point of intersection of LB and LR can be determined in $O(T(n))$ time using $O(S(n))$ space.*

Proof. As earlier, we can assume that the points of intersection of the dual lines R' with the vertical line V are available in a read-only array. Now, apply the median finding algorithm (see Table 1) to compute the middle-most point z_r among the points of intersection of the members of R' and V. Similarly, compute the middle-most point z_b among the points of intersection of the members of B' and V. Now, since LR is monotone decreasing and LB is monotone increasing, if $y(z_r) > y(z_b)$ (resp. $y(z_r) < y(z_b)$) then the point of intersection of LR and LB is to the right (resp. left) of the vertical line V. The time and extra space complexity results follow from that of the median finding algorithm. □

Lemma 8 *Given the points R and B in two read-only arrays and a line ℓ with positive slope, the side of ℓ containing the point of intersection θ of LB and LR can be determined in $O(T(n))$ time using $O(S(n))$ space.*

Proof. Consider the lines in R (having negative slope). The point of intersection α of the monotone decreasing polychain LR with the line ℓ corresponds to the $\frac{n}{2}$-th intersection point among the points of intersection of ℓ and the lines in R. This can be computed in $O(T(n))$ time using $O(S(n))$ space. Let us now draw a vertical line V_α at the point α, and compute the point of intersection β of V_α and the monotone increasing polychain LB in $O(T(n))$ time using $O(S(n))$ space (see Lemma 7). The points β and θ will lie in the same side of ℓ [13]. Thus, the result follows. □

Based on the results in Lemmas 7 and 8, we now describe the execution steps of an iteration of the prune and search algorithm for computing the ham-sandwich cut line L.

Step 1: Find μ the median slope of all remaining lines in R' and B' (independently of their color).

Step 2: Arbitrarily pair-up lines having slope greater and less than μ. Let I be the set of all these intersections.

Step 3: Compute the median of the x-coordinates of the points in I; let it correspond to a point of intersection $u \in I$. The vertical line V_u bisects the points of intersection in I.

Step 4: Using Lemma 7, compute the side of the line V_u that contains the intersection of the median levels LR and LB. (suppose the intersection is to the right).

Step 5: Find a horizontal line H_v that bisects the points of I to the left of V_u.

Step 6: Use Lemma 8 to compute the side of H_v containing the intersection of the median levels LR and LB. Let it be in the top-right quadrant defined by V_u and H_v.

Step 7: For each point of I lying in the bottom-right quadrant, prune the line having slope less than μ since it can not enter in the top-right quadrant. Thus, we can prune at least $\frac{1}{8}$ fraction of the remaining lines for the next iteration.

The process terminates when either z^* is the intersection point of V_u and H_v, or a constant number of lines remain. In the second case, we enumerate the intersection point of LB and LR, computing these chains explicitly with the not-pruned points.

Lemma 9 *Given two sets of vertically separable points R and B ($|R|+|B| = n$) in two read-only arrays, one can compute a straight-line L that equi-partitions both R and B simultaneously in $O(T(n)\log n)$ time using $O(S(n))$ extra space.*

Proof. The time complexity of each iteration of the algorithm is $O(T(n))$. Since, in each iteration a constant fraction of the existing lines are pruned, the time complexity result follows.

In order to justify the space complexity result, we have to show that we can store the summery of pruning information in $O(1)$ space. Note that, the feasible region of the solution point in the dual space is an axis-parallel rectangle (bounded or unbounded), which can be stored in $O(1)$ space. The other information is the value of μ. The range of values of μ ($\mu < 0$) for which the dual (red) lines will be pruned can be kept in 2 extra locations. The same is true for $\mu > 0$ (blue lines). Thus, the extra space complexity is dominated by that of the median finding algorithm. □

3.3 Complexity analysis

Theorem 10 *Given a point set P and an integer k, the proposed algorithm reports the existence of a k-centerpoint of P in $O(T(n^2)\log^2 n)$ time using $S(n^2)$ extra space.*

Proof. Let m be the number of vertices of $\mathcal{A}(P')$. We compute the leftmost and rightmost vertices u_0, u_m in $O(n^2)$ time. Let (a, b) denote the range of x-coordinates in which both the end-points of the desired bridge is expected to lie. Initially, $a = u_0$ and $b = u_m$ is set. In each iteration, different steps are listed below along with the time complexities:

Step 1: Computation of the vertex $u \in \mathcal{A}(P')$ lying in $[a, b]$, to define V_u, which takes $T(n^2)$ time,

Step 2: Computation of edge $e \in \lambda_k$ that intersects V_u, which takes $O(T(n^2)\log n)$ time (see Lemma 9).

Step 3: Testing of whether e is a tangent of $\lambda_{(n-k)}$ takes $O(n^2)$ time (see Lemma 2).

Thus, the time complexity of each iteration is $O(T(n^2)\log n)$. After an iteration, if the test in Step 3 fails, then we update a or b with u as mentioned in Subsection 3.1, and execute the same steps with the vertices of $\mathcal{A}(P')$ that lie in the updated x-interval $[a, b]$. Since at least half of the vertices lie outside the updated $[a, b]$, the number of such iterations is $O(\log n)$. Now, after processing λ_k, if the decision can not be taken, then the same process is repeated with $\lambda_{(n-k)}$, and the time complexity remains unchanged. Finally, one may check whether a line is a tangent of both λ_k and $\lambda_{(n-k)}$, which takes $O(n^2)$ time. Thus, the overall time complexity is $O(T(n^2)\log^2 n)$. □

References

[1] L. Barba, M. Korman, S. Langerman, K. Sadakane and R. I. Silveira. Space-Time Trade-offs for Stack-Based Algorithms. *STACS*, 2013.

[2] T. M. Chan, Comparison-based time-space lower bounds for selection. *ACM Transactions on Algorithms*, vol. 6(2), 2010.

[3] T. M. Chan, J. I. Munro and V. Raman. Faster, space-efficient selection algorithms in read-only memory for integers. *ISAAC*, pp. 405-412, 2013.

[4] T. M. Chan, J. Ian Munro, V. Raman. Selection and Sorting in the "Restore" Model. *SODA*, pp. 995-1004, 2014.

[5] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, Jan. 1987.

[6] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 154–166, New York, NY, USA, 1984. ACM.

[7] T. K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[8] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer.

[9] A. Elmasry, D. D. Juhl, J. Katajainen and S. Rao Satti. Selection from read-only memory with limited work space. in Proc. *COCOON*, pp. 147157, 2013.

[10] G. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System Sciences*, vol. 34(1), pp. 1926, 1987.

[11] S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete & Computational Geometry*, 12:291–312, 1994.

[12] D. G. Kirkpatrick and R. Seidel, The Ultimate Planar Convex Hull Algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

[13] J. Matousek. Approximations and optimal geometric divide-an-conquer. *J. Comput. Syst. Sci.*, 50(2):203–208, 1995.

[14] J. Matousek. Computing the center of planar point sets, 2000.

[15] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, Oct. 1983.

[16] N. Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 6(3):430–433, 1985.

[17] J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, vol. 12, pp. 315-323, 1980.

[18] J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996.

[19] N. Naor and M. Sharir. Computing a point in the center of a point set in three dimensions. *CCCG*, 1990.

[20] V. Raman and S. Ramnath. Improved upper bounds for time-space tradeoffs for selection with limited storage. In *SWAT*, pages 131–142, 1998.