The correspondence between Donald E. Knuth and Peter van Emde Boas on priority deques during the spring of 1977

March 29 1977 Knuth mailed a copy of his classroom note on the van Emde Boas construction of priority deques to Peter van Emde Boas in reply to a preprint 77-05 of the University of Amsterdam. This report contains a solution to the problem of the super-linear space consumption of the priority deque as originally described in the paper presented at FOCS 1975.

Aside of the fact that this note is the source of the well known quotation by Knuth on program correctness, it is the first document which describes these priority deques using the top-down recursive format which is common today.

This recursive approach which was suggested in the FOCS 1975 paper was not explored in the original publications since it would involve the use of multiplicative machine instructions for address calculations which, during this period were considered to be inadmissible for the RAM model with uniform time measure.

These issues are discussed in the two letters which are included in the facsimile edition: the cover letter by Knuth and the reply by van Emde Boas. An more extensive discussion of the historical context can be found in [6]. Time has arrived to make this document accessible on the Web.

References

[1] D.E. Knuth, *Notes on the van Emde Boas construction of priority deques: an instructive use of recursion*, Classroom notes Stanford University, March 1977.

[2] P. van Emde Boas, *An O(nloglogn) On-line Algorithm for the Insert-Extract Min problem*, Report Cornell University Dept. of Computer Science TR 74-221, december 1974; online accessible at https://dspace.library.cornell.edu/bitstream/1813/6060/1/74-221.pdf .

Figure 1: the cover of Knuth's letter

[3] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time*, Proc. IEEE FOCS 16, Berkeley, Oct 1975, pp. 75–84, preprint: Report Mathematical Centre Amsterdam, MC-ZW-55-75.

[4] P. van Emde Boas, R. Kaas & E. Zijlstra, *Design and implementation of an efficient priority queue*, Math. Syst. Theory 10 (1977) 99–128, preprint: Report Mathematical Centre Amsterdam, MC-ZW-60-75 .

[5] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time and linear space*, Inf. Proc. Letters 6 (1977) 80–82, preprint: Report dept. of Mathematics, University of Amsterdam 77-05.

[6] P. van Emde Boas, *Thirty nine years of stratified trees*, Proc. ISCIM 2013, Tirana, Albania.

Notes on the van Emde Boas construction of priority deques: An instructive use of recursion. D. Knuth, March 1977.

A. The problem is to represent subsets S of $\{0,1,\ldots,n-1\}$, and to implement the following operations when s is a pointer to such a representation:

s.card:	the cardinality of S
s.min:	the smallest element of S (undefined if S is empty)
s.max:	the largest element of S (undefined if S is empty)
s.insert(x):	insert x into S (undefined if $x \in S$)
s.delete(x):	delete x from S (undefined if $x \notin S$)
s.successor(x):	the smallest element $\geq x$ in S (undefined if $x >$ s.max) .

A predecessor operation could be defined similarly.

If n is small, we can use bit manipulation techniques in a standard way. For larger n, let $n = n_1 n_2$, and for $0 \leq x < n$ let $q(x) = \lfloor x/n_2 \rfloor$, $r(x) = x \bmod n_2$. We can build a representation from n_1 priority deques of order n_2, plus an additional priority deque of order n_1 to keep track of which of the others are nonempty. Thus the universe ismade up of n_1 "galaxies" of smaller universes. Here is the recursive construction, using SIMULA-like notation, emphasizing ease of verification rather than efficiency.

```
class pd;
begin internal integer size, least, greatest; comment ||S||, min(S), and max(S);
   internal ref(pd) T; comment a priority deque of order n1 for the nonempty
      galaxies;
   internal ref(pd) array g[0:n1-1]; comment priority deques for the galaxies;
   comment initially size = 0, least = ∞, greatest = -∞, T points to an
      initialized pd of order n1 and g[0]...g[n1-1] each point to disjoint
      initialized pd's of order n2;
   external integer procedure card; return (size);
   external integer procedure min; return (least);
   external integer procedure max; return (greatest);
   external procedure insert (integer x);
```


Figure 2: page 1 of classroom note

```
begin size ← size+1; if x < least then least ← x; if x > greatest
   then greatest ← x;
if g[q(x)].card = 0 then T.insert(q(x));
g[q(x)].insert(r(x));
end;
external procedure delete (integer x);
begin size ← size - 1;
   g[q(x)].delete(r(x));
   if g[q(x)].card = 0 then T.delete(q(x));
   if size = 0 then (least ← ∞; greatest ← -∞)
      else (least ← g[T.min].min + n2 x T.min;
            greatest ← g[T.max].max + n2 x T.max)
end;
external integer procedure successor (integer x);
   if r(x) > g[q(x)].max then return (g[T.successor(q(x))].min
                                        + n2 x T.successor(q(x)))
   else return (g[q(x)].successor(r(x)) + n2 x q(x));
end pd.
```

B. The running time.

The time for card, min, max is $O(1)$.
For insert, $T(n) = O(1) + T(n_2)$ + if trivial n_2 insertion then $T(n_1)$.
For delete, $T(n) = O(1) + T(n_2)$ + if trivial n_2 deletion then $T(n_1)$.
For successor, $T(n) \le O(1) + \max(T(n_1), T(n_2))$.

Thus we get the best performance by taking $n_1 \approx n_2 \approx \sqrt{n}$. The solution to the recurrence $T(n) = 2T(\sqrt{n}) + O(1)$ has $T(n) \asymp \log n$ while the solution to $T(n) = T(\sqrt{n}) + O(1)$ has $T(n) \asymp \log \log n$, hence for best asymptotic growth we should try to improve the algorithm. [For small n the difference is probably negligible and the "improved" algorithm might even run slower if we aren't careful ... this should be studied on real machines, but let us Think Big for now.] The desired improvement is to ensure that trivial insertions and deletions always take $O(1)$ time. The tricky thing is that trivial deletions don't have time to clean up the data structure, and trivial insertions haven't the time to reinitialize it. But we can solve the problem by making the work incremental in the

Figure 3: page 2 of classroom note

following way: Whenever size $\leq$ 1 , the galaxies are not used and all the subsidiary pd's will be empty.

C. The revised program, which has O(log log n) running time for all operations.

```
class pd
begin ... same as before ...;
external procedure insert (integer x);
if size = 0 then (size ← 1; least ← greatest ← x)
else begin if size = 1 then (T.insert(q(least)); g[q(least)].insert(r(least)));
   if g[q(x)].card = 0 then T.insert(q(x));
   g[q(x)].insert(r(x));
   size ← size + 1;
   if x < least then least ← x
   else if x > greatest then greatest ← x;
   end;
external procedure delete (integer x);
begin size ← size - 1;
if size = 0 then (least ← ∞; greatest ← -∞)
else begin g[q(x)].delete(r(x));
   if g[q(x)].card = 0 then T.delete(q(x));
   if x = least then least ← g[T.min].min + n2 x T.min
   else if x = greatest then greatest ← g[T.max].max + n2 x T.max;
   if size = 1 then (T.delete(q(least)); g[q(least)].delete(r(least)));
end;
external integer procedure successor (integer x);
begin if size ≤ 1 then return (greatest)
   else if r(x) ... as before...;
end;
end pd.
```


Figure 4: page 3 of classroom note

D. Elimination of class pointers, and storage allocation.

The construction will never go many levels before exceeding the size of the real universe. For example, if we have a machine with 32-bit words, the bottom level pd routine will use the computer's built-in operations, the next level takes n up to $32 \cdot 32 = 2^{10}$, the next level already takes us to $n = 2^{20}$ (greater than a million), and the next level makes n greater than the largest existing computer memories. Let's suppose $n = 2^{20}$. We will have one pd of height 2 (it has three local variables: size, least, greatest, of 20 bits each), $2^{10}+1$ pd's of height 1 (each with three local variables of 10 bits each), and $(2^5+1)(2^{10}+1)$ pd's of height 0 (each with three local variables of 5 bits each and one 32-bit table). Rounding up to 32, 16, 8-bit variables, we see that the memory requirements on the IBM 360 would be $3 \cdot 2^2 + 3 \cdot (2^{10}+1)2^1 + 7 \cdot (2^5+1)(2^{10}+1)2^0) = 242,937$ bytes. Note that 2^{20} bits is 131072 bytes, so the overhead is not substantial.

But let's assume a 16-bit computer word, so the calculations are cleaner and nicer. Then $n = 2^{16} = 65536$, a reasonable size for applications. For convenience we will be generous and allow 16 bits for all local variables at all levels. This wastes memory by a factor of about 2, but it speeds up the programs since we can use x instead of r(x) in the galaxies. Let us use three arrays size, least, greatest, B[0:4626] for these tables; here B holds the 16-bit codes for the bottom level. A level-1 structure whose variables begin at location ℓ will have 17 bottom-level structures whose variables begin at $\ell+1, \ldots, \ell+17$ respectively. The following implementation of insert indicates how the rest of the implementation can be carried out. The notation $x \rightarrow k$ means x shifted right k, i.e., $\lfloor x/2^k \rfloor$.

Figure 5: page 4 of classroom note

```
procedure insert0 (integer x);
if size[0] = 0 then (size[0] ← 1; least[0] ← greatest[0] ← x)
else begin if size[0] = 1 then (insert1(least[0]↱8,1);
          insert 1(least[0], 18 x (least[0]↱8) + 19));
   if size[18 x (x↱8) + 19] = 0 then insert1(x↱8,1);
   insert1(x, 18 x (x↱8) + 19);
   size[0] ← size[0] + 1;
   if x < least[0] then least[0] ← x
   else if x > greatest[0] then greatest[0] ← x;
   end;
procedure insert1 (integer x, ℓ);
if size[ℓ] = 0 then (size[ℓ] ← 1; least[ℓ] ← greatest[ℓ] ← x)
else begin if size[ℓ] = 1 then (insert2(least[ℓ]↱4, ℓ+1);
          insert2(least[ℓ], (least[ℓ]↱4) mod 16 + ℓ+2));
   if size[(x↱4) mod 16 + ℓ+2] = 0 then insert2(x↱4, ℓ+1);
   insert2(x, (x↱4) mod 16 + ℓ+2);
   size[ℓ] ← size[ℓ]+1;
   if x < least[ℓ] then least[ℓ] ← x
   else if x > greatest[ℓ] then greatest[ℓ] ← x;
   end;
procedure insert2 (integer x, ℓ)
begin B[ℓ] ← B[ℓ] ∨ (2 ↑ (x mod 16));
   size[ℓ] ← size[ℓ]+1;
   if x < least[ℓ] then least[ℓ] ← x
   else if x > greatest[ℓ] then greatest[ℓ] ← x;
   end;
```

The implementation of deletion would be similar. It is safe to use 0 and $2^{16}-1$ for $-\infty$ and $+\infty$.

Beware of bugs in the above code; I have only proved it correct, not tried it.

Figure 6: page 5 of classroom note

STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

COMPUTER SCIENCE DEPARTMENT

Telephone:
415-321-2300

March 29, 1977

Dr. P. van Emde Boas
Instituut voor Toepassingen der Wiskunde
University of Amsterdam
Roetersstraat 15
Amsterdam-C, Netherlands

Dear Sir,

Thank you very much for report 77-05 which I recently received.

Since I have been working mostly on other parts of my book, I have not yet found the chance to study your papers as much as I should, but I have some questions and perhaps you can help put me on the right track.

In your paper about the PASCAL implementation, you give the space requirements as $O(n \log\log n)$, but as nearly as I can tell from your proof this means $O(n \log\log n)$ pointers rather than bits. Thus, an additional factor of $\log n$ creeps in. On the other hand, I think your approach yields $O(\log\log n)$ processing time and $O(n)$ space in bits (on some appropriate machine model, e.g. Pratt and Stockmeyer in JCSS) if you return to the idea expressed in your Cornell report that arithmetic operations be used to get around in the tree. Here is my argument, and I would appreciate your comments if you have time.

Let $f(n)$ be any function which is $O(n/\log n)$. Then the solution to the recurrence

$$S(n) = f(n) + S(\sqrt{n}) + \sqrt{n}\, S(\sqrt{n}) \quad , \quad S(1) = f(1)$$

is linear in n. Proof: Choose n_0 and β so that $f(n) \le \beta\left(\frac{n}{\log n} + \frac{2\sqrt{n}}{\log n} - \sqrt{n} \right)$ for all $n \ge n_0$. Then choose $\alpha \ge \beta$ so that $S(n) \le \alpha(n - n/(\log n))$ for $1 < n \le n_0$. We can now prove that $S(n) \le \alpha(n - n/\log n)$ for all n, by induction:

$$S(n) = f(n) + S(\sqrt{n}) + \sqrt{n}\, S(\sqrt{n}) \le f(n) + \alpha\left(\sqrt{n} - \frac{2\sqrt{n}}{\log n} + n - \frac{2n}{\log n} \right)$$
$$\le \alpha\left(n - \frac{n}{\log n} \right) - (\alpha-\beta)\left(\frac{n}{\log n} + \frac{2\sqrt{n}}{\log n} - \sqrt{n} \right) \le \alpha\left(n - \frac{n}{\log n} \right) .$$

Thus, the storage requirements for a recursive universe-cluster-galaxy scheme will be linear if the storage $f(n)$ at a given level of recursion

Figure 7: first page of Knuth's letter

is not too large. I have attached to this letter my current favorite way to implement your scheme. (I use SIMULA classes instead of PASCAL; I think SIMULA has a better approach to recursive data structures, although I probably abuse its syntax.) Since the membership test is not used in the other operations, I left it out; clearly it can also be done in O(log log n) steps. Curiously, there seems to be no need for the concepts of rank, canonical subtree, branch points, etc. when you view the procedure "from the top down". Thus everything is simplified including the job of proving correctness. If nobody has received the $10 yet, I think this program is worth $5 anyway.

I will of course be referring to your method in my "volume 4", and for this purpose I need to know your full name including any middle names; this will appear in the index. (I have been alphabetizing Dutch names like de Bruijn, van Emden, van Lint, etc. under the letters "d" and "v", as is customary in America; but if you would like me to list you under E as well please let me know!)

Cordially,

Donald E. Knuth/pw

Donald E. Knuth
Professor

DEK/pw

Encl.

cc: J. Hopcroft
R. Tarjan

Figure 8: second page of Knuth's letter

Foundation Mathematisch Centrum

2e Boerhaavestraat 49 Amsterdam-1005

MC

Phone (020) 94 72 72
Telex 12571
Bankers: Amsterdam-Rotterdam Bank NV
Branch office Sarphatistraat, Amsterdam

Your reference
Dated March 29 1977
Our reference
Date April 15 1977

Prof dr D.E.Knuth
Computer science department
Stanford University
Stanford, CA 94305

Dear Sir

Thanks for your letter of march 29. The problems concerning the space and time requirements of my priority deque scheme (thank you for the name) have kept my attention since the time the scheme was designed originally in 74. The recent report UVA 05-77 is the last offspring of this research.
Before answering your questions I like to mention that the final version of the FOCS 16 paper will appear on short time in the forthcomming issue of Mathematical Systems Theory. This paper contains the complete PASCAL program and in it it is stated explicitly that the storage requirements are expressed in RAM words and not in bits. On the other hand it can be read from the PASCAL programs that no other arithmetic operations are used than additions and subtractions, this way legalizing the use of the uniform RAM time measure (one step per instruction executed).
The idea of using arithmetic operations for computing adresses within the tree was originally rejected since the needed instructions essentially amount to decomposing a given bit string into two equal parts, which instruction is not available on a RAM, and which costs, if programmed , logarithmic time. The instruction is available on a vector machine like proposed by Pratt and Stockmeyer, but their results show that their machine, being not polynomially equivalent to Turing machines unless P = PSPACE, should be rejected as a reasonable machine model. As a consequence,on base of the rules of the game as Hopcroft has thought them to me, I was forced to loose my time efficiency on behalf of the adress computations, or to use the intricate collection of pointers which ammounts to an order loglogn overhead factor in the space requirements plus an initialization time of order n.loglogn. This way the structure has been described in the paper. The recent report 05-77 shows that the loglogn factor can be eliminated by use of one more cluster-galaxy decomposition using two different priority deque schemes.

Figure 9: first page of van Emde Boas' letter

If we agree on some restricted vector machine for which the Pratt-Stockmeyer result no longer holds, but which is polynomially equivalent to Turing machines clearly an order n.loglogn -time, order n space structure can be obtained (assuming that the semingly innocent instruction of breaking a bit string into equal parts by itself does not yield the full power of parallellism as is the case for the complete vector machine). I have not considered whether I should introduce such a machine model for this specific application. Moreover the problems would return when one likes to use the structure for storing O(n) distinct items since in this case space O(n.loglogn) is needed anyhow.

I agree that by use of a recursive data structure both the algorithms and the correctness proofs become easier. I have realised so at the time the $10.-- prise was offered for a correctness proof of the algorithms as given for the nonrecursive structure. I have realised however as well that the structure as described in the FOCS 16 or MST paper can not be obtained by unwinding the recursion from a structure as given in your SIMULA text, because of absence of separate treatment of left- and right-hand subtrees. Consequently the FOCS 16 structure is really too complex, and I am looking forwards of putting together a nonrecursive simplified version for which I have to overcome several two year old prejudices. As explained above I do not consider your note a legitimate solution for being awarded the still unclaimed $10.-- prize, but under separate cover I will mail you some piece of typical Dutch furniture, approximating the amount claimed.

For your bibliography in "Volume 4" I like to mention that my complete name reads : Peter van Emde Boas . This name which involves a single first name and no middle name has caused great problems to many Americans before. I have considered declaring a middle name "vanEmde" but I consider this to be an illegal action. Alphabetizing it under the V is OK to me.

In case you have been informed on applications of my scheme I would like to hear about it. Most applications of priority queues either involve real valued items, or a number of items which is much smaller than the n involved and for both types of applications the scheme can not be used.

Sincerely yours

Peter van Emde Boas

Figure 10: second page of van Emde Boas' letter