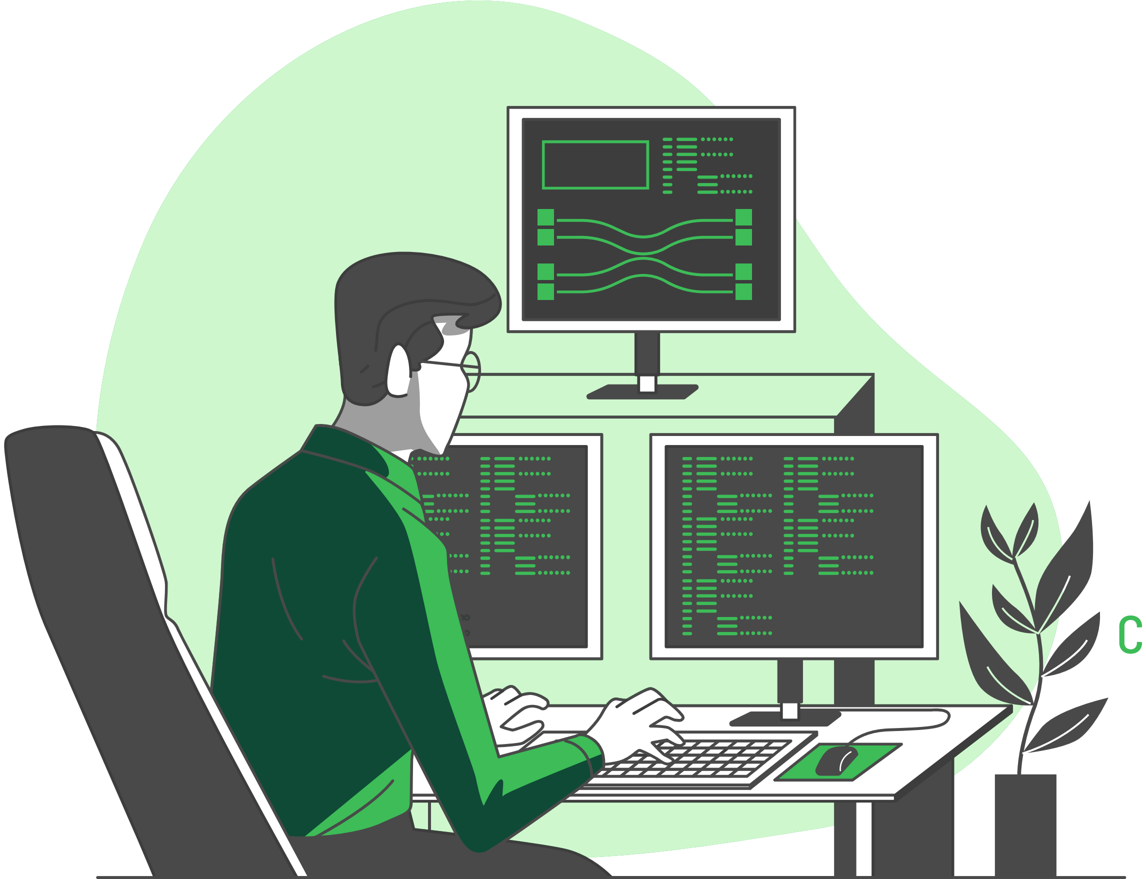
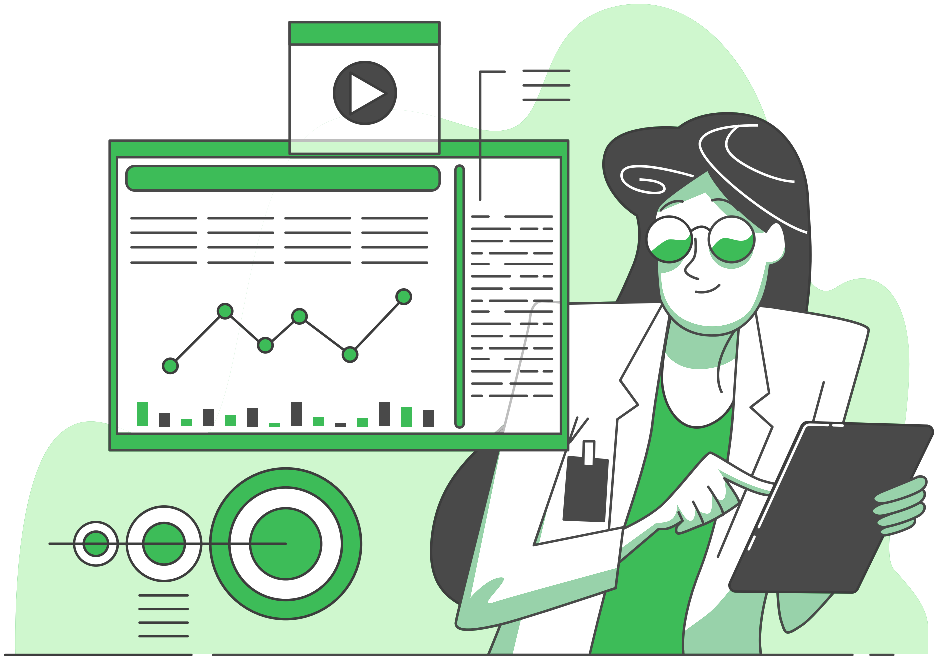


Heapify

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents



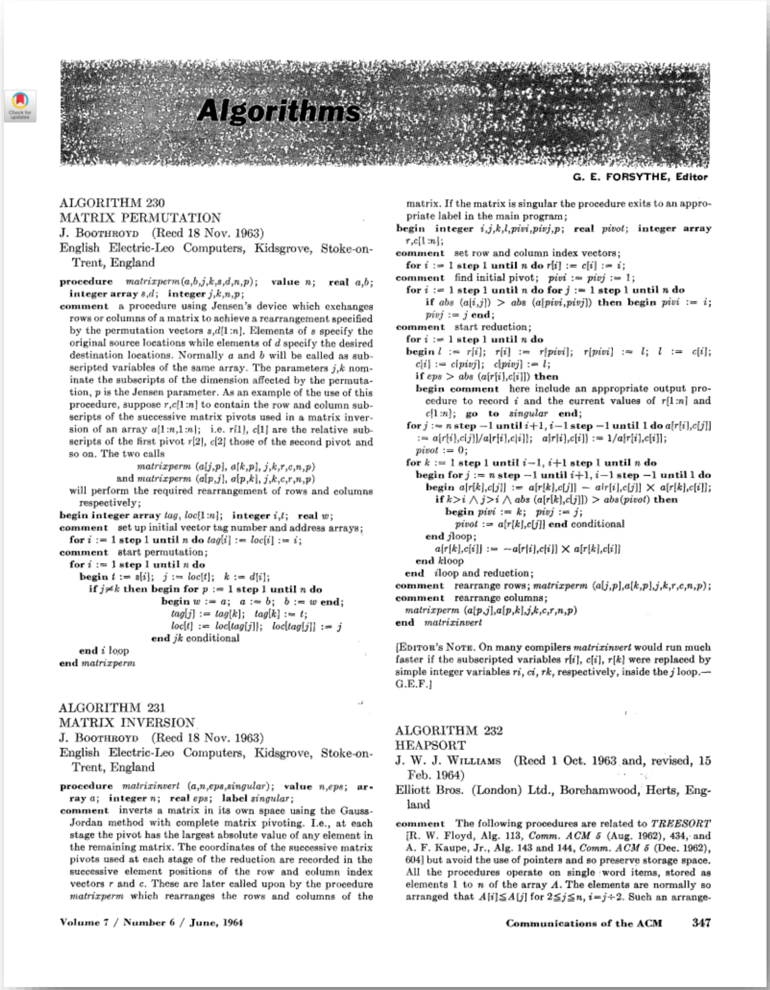
01
Heapify

Transform an array into a binary heap

Think about this

1. A single insert into a binary heap is $O(\log_2(n))$.
2. Inserting n items is $O(n \log_2(n))$.
3. If we have an array with n items to be inserted in a binary heap, can we build the binary heap better than $O(n \log_2(n))$?

"Algorithm 232 – Heapsort“, J. W. J. Williams, "Communications of the ACM“, 1964



Algorithms

G. E. FORSYTHE, Editor

ALGORITHM 230
MATRIX PERMUTATION
J. Boothroyd (Recd 18 Nov. 1963)
English Electric-Leo Computers, Kidsgrove, Stoke-on-Trent, England

```
procedure matrixperm(a,b,j,k,s,d,n,p); value n; real a,b;
  integer array s,d; integer j,k,n,p;
comment a procedure using Jensen's device which exchanges
  rows or columns of a matrix to achieve a rearrangement specified
  by the permutation vectors s,d[1:n]. Elements of s specify the
  original source locations while elements of d specify the desired
  destination locations. Normally a and b will be called as sub-
  scripted variables of the same array. The parameters j,k nom-
  inate the subscripts of the dimension affected by the permuta-
  tion, p is the Jensen parameter. As an example of the use of this
  procedure, suppose r,c[1:n] to contain the row and column sub-
  scripts of the successive matrix pivots used in a matrix inver-
  sion of an array a[1:n,1:n]; i.e. r[1], c[1] are the relative sub-
  scripts of the first pivot r[2], c[2] those of the second pivot and
  so on. The two calls
              matrixperm (a[j,p], a[k,p], j,k,r,c,n,p)
          and matrixperm (a[p,j], a[p,k], j,k,c,r,n,p)
  will perform the required rearrangement of rows and columns
    respectively;
begin integer array tag, loc[1:n]; integer i,t; real w;
comment set up initial vector tag number and address arrays;
  for i := 1 step 1 until n do tag[i] := loc[i] := i;
comment start permutation;
  for i := 1 step 1 until n do
    begin t := s[i]; j := loc[t]; k := d[i];
      if j≠k then begin for p := 1 step 1 until n do
                   begin w := a; a := b; b := w end;
                     tag[j] := tag[k]; tag[k] := t;
                     loc[t] := loc[tag[j]]; loc[tag[j]] := j
                 end jk conditional
    end i loop
end matrixperm
```

ALGORITHM 231
MATRIX INVERSION
J. Boothroyd (Recd 18 Nov. 1963)
English Electric-Leo Computers, Kidsgrove, Stoke-on-Trent, England

```
procedure matrixinvert (a,n,eps,singular); value n,eps; ar-
  ray a; integer n; real eps; label singular;
comment inverts a matrix in its own space using the Gauss-
  Jordan method with complete matrix pivoting. I.e., at each
  stage the pivot has the largest absolute value of any element in
  the remaining matrix. The coordinates of the successive matrix
  pivots used at each stage of the reduction are recorded in the
  successive element positions of the row and column index
  vectors r and c. These are later called upon by the procedure
  matrixperm which rearranges the rows and columns of the
  matrix. If the matrix is singular the procedure exits to an appro-
  priate label in the main program;
begin integer i,j,k,l,pivi,pivj,p; real pivot; integer array
  r,c[1:n];
comment set row and column index vectors;
  for i := 1 step 1 until n do r[i] := c[i] := i;
comment find initial pivot; pivi := pivj := 1;
  for i := 1 step 1 until n do for j := 1 step 1 until n do
    if abs (a[i,j]) > abs (a[pivi,pivj]) then begin pivi := i;
    pivj := j end;
comment start reduction;
  for i := 1 step 1 until n do
  begin l := r[i]; r[i] := r[pivi]; r[pivi] := l; l := c[i];
    c[i] := c[pivj]; c[pivj] := l;
    if eps > abs (a[r[i],c[i]]) then
    begin comment here include an appropriate output pro-
      cedure to record i and the current values of r[1:n] and
      c[1:n]; go to singular end;
  for j := n step -1 until i+1, i-1 step -1 until 1 do a[r[i],c[j]]
    := a[r[i],c[j]]/a[r[i],c[i]]; a[r[i],c[i]] := 1/a[r[i],c[i]];
    pivot := 0;
  for k := 1 step 1 until i-1, i+1 step 1 until n do
    begin for j := n step -1 until i+1, i-1 step -1 until 1 do
      begin a[r[k],c[j]] := a[r[k],c[j]] - a[r[i],c[j]] × a[r[k],c[i]];
        if k>i ∧ j>i ∧ abs (a[r[k],c[j]]) > abs(pivot) then
          begin pivi := k; pivj := j;
            pivot := a[r[k],c[j]] end conditional
      end jloop;
        a[r[k],c[i]] := -a[r[i],c[i]] × a[r[k],c[i]]
    end kloop
  end iloop and reduction;
comment rearrange rows; matrixperm (a[j,p],a[k,p],j,k,r,c,n,p);
comment rearrange columns;
  matrixperm (a[p,j],a[p,k],j,k,c,r,n,p)
end matrixinvert
```

[Editor's Note. On many compilers *matrixinvert* would run much faster if the subscripted variables *r*[*i*], *c*[*i*], *r*[*k*] were replaced by simple integer variables *ri*, *ci*, *rk*, respectively, inside the *j* loop.—G.E.F.]

ALGORITHM 232
HEAPSORT
J. W. J. Williams (Recd 1 Oct. 1963 and, revised, 15 Feb. 1964)
Elliott Bros. (London) Ltd., Borehamwood, Herts, England

```
comment The following procedures are related to TREESORT
  [R. W. Floyd, Alg. 113, Comm. ACM 5 (Aug. 1962), 434, and
  A. F. Kaupe, Jr., Alg. 143 and 144, Comm. ACM 5 (Dec. 1962),
  604] but avoid the use of pointers and so preserve storage space.
  All the procedures operate on single word items, stored as
  elements 1 to n of the array A. The elements are normally so
  arranged that A[i]≤A[j] for 2≤j≤n, i=j÷2. Such an arrange-
```


https://dl.acm.org/doi/10.1145/512274.512284

Max Child and Sift Down

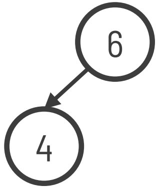
```
algorith maxchild(A:array, n:ℤ≥0, i:ℤ≥0) → ℤ≥0
   lc ← leftchild(i)
   if lc >= n then
      return n
   end if
   rc ← rightchild(i)
   if rc >= n then
      return lc
   end if
   if A[lc] > A[rc] then
      return lc
   else
      return rc
   end if
end algorithm
```

```
algorith siftdown(A:array, n:ℤ≥0, i:ℤ≥0)
   m ← maxchild(A, n, i)
   while m < n and A[i] < A[m] do
      swap(A, i, m)
      i ← m
      m ← maxchild(A, n, i)
   end while
end algorithm
```

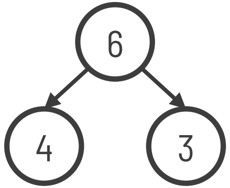
Build heap: 4, 6, 3, 5, 7, 1

Traditional approach (inserting one item at a time)

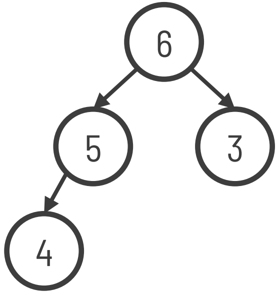
0	1	2	3	4	5
4					



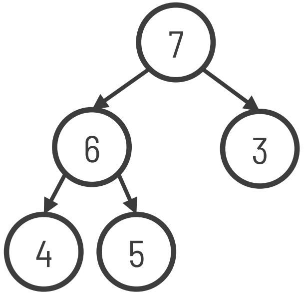
0	1	2	3	4	5
6	4				



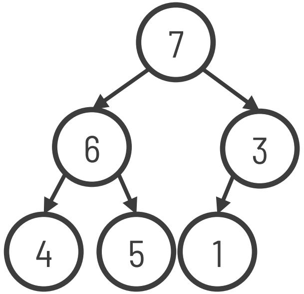
0	1	2	3	4	5
6	4	3			



0	1	2	3	4	5
6	5	3	4		



0	1	2	3	4	5
7	6	3	4	5	



0	1	2	3	4	5
7	6	3	4	5	1

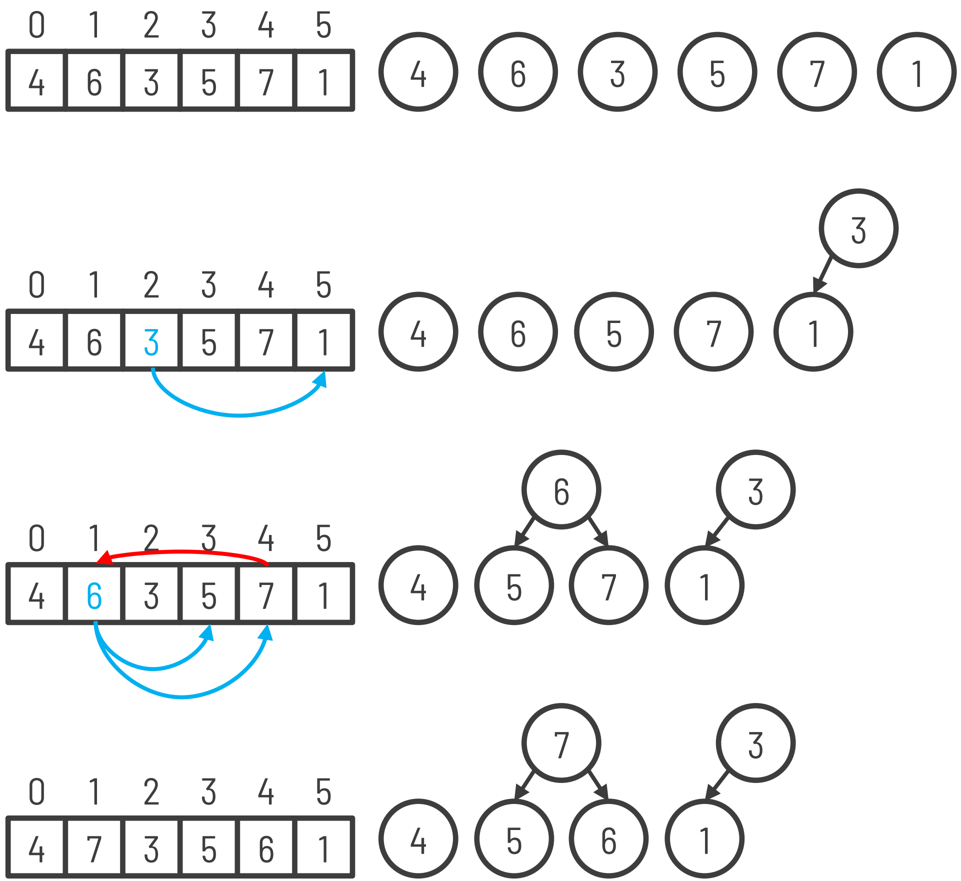
Heapify

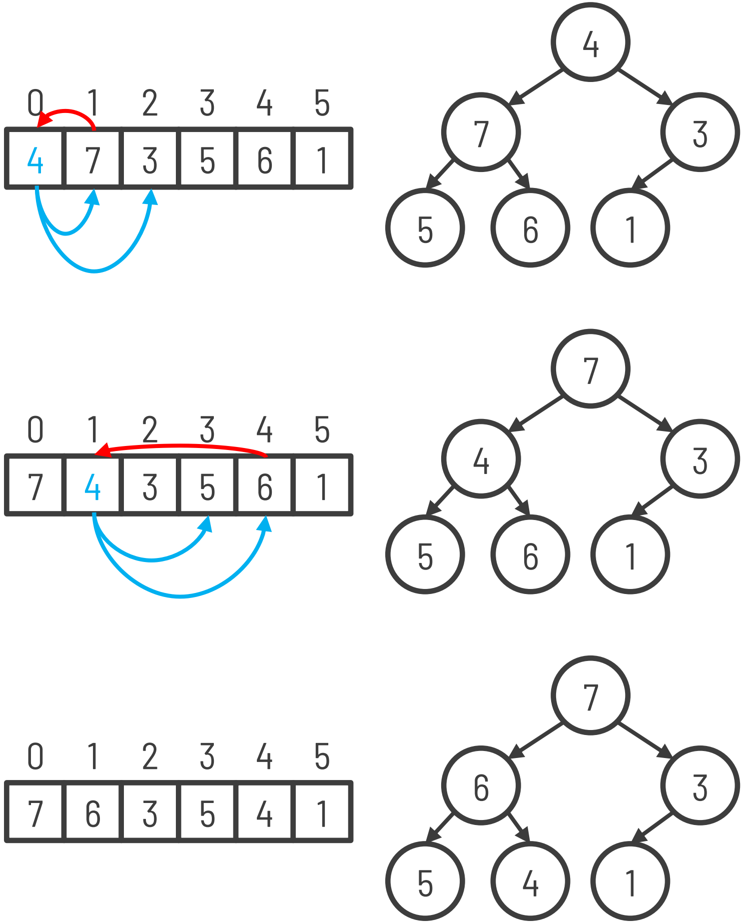
```
algorith heapify(A:array, n:ℤ≥0)
   for i from floor(n/2) – 1 to 0 by -1 do
      siftdown(A, n, i)
   end for
end algorithm
```

It is convenient to have n (i.e., the size of the array) as an argument for heapify. It will come in handy later when discussing Heapsort.

Build heap: [4, 6, 3, 5, 7, 1]

heapify (transform an array into a binary heap)





Heapify Remarks

- Also known as **Bottom-Up** (start at the leaves).
- Considers the items between indices 0 and $\left\lfloor \frac{\boldsymbol{n}}{\mathbf{2}} \right\rfloor - \mathbf{1}$
- Call **Sink/Swim/Sift Down** on each node.
- Heapify builds the binary heap in $O(n)$.

Done, Sort Of

Do you have any questions?

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories