

CS420 Introduction to Algorithms:

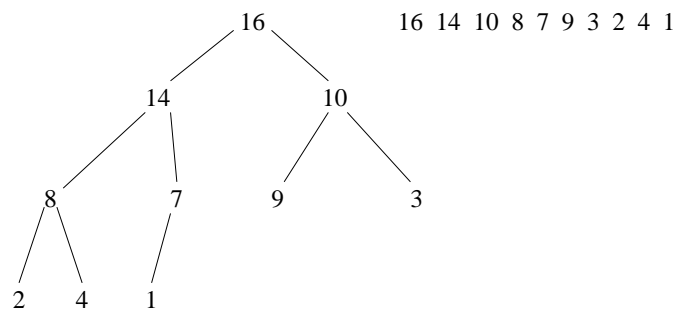
Darrell Whitley
Colorado State University
Fort Collins, CO 80523

Slide Lecture 1 -1

HEAPSORT

heap: array representation of a binary tree

Tree completely filled except on lowest level, which is filled
from left to right



Slide Lecture 1 -2

$$\text{parent}[i] = \lfloor i/2 \rfloor$$

$$\text{leftchild}[i] = 2i$$

$$\text{rightchild}[i] = 2i + 1$$

HEAP Property

$$\text{Array}[\text{parent}(i)] \geq \text{Array}[i]$$

$$\text{Array}[2] = 14 \geq 8 = \text{Array}[4]$$

With N elements, height of heap is $(\lg N)$

Slide Lecture 1 -3

HEAPIFY (Array, i)

Preconditions: trees at left(i) and right(i) are heaps

Postconditions: tree at i is heap

Idea: bubble Array[i] down until it satisfies the heap property.

I.E., swap Array(i) with max left and right (denote MLR) as long as Array(i) < MLR.

Slide Lecture 1 -4

```

HEAPIFY (Array, i)

L := left(i); R := right(i);

if L <= N and Array(L) > Array(i)
    then max := L else max := i;

if R <= N and Array(R) > Array(max)
    then max := R;

if max != i then {SWAP(i,max); HEAPIFY(Array, max)}
end HEAPIFY

```

Slide Lecture 1 -5

Number of swaps performed by HEAPIFY(Array,i) is at most the distance from root to leaf = $O(\lg N)$.

Building a heap out of an array[1..n]
observation all elements $A[\lfloor n/2 \rfloor + 1 \dots n]$ are
leaves \rightarrow heaps.

rest of A is built into a heap by bottom up application of
HEAPIFY.

```

BUILDHEAP (Array)

for i := Floor(N/2) downto 1
    HEAPIFY (Array, i)

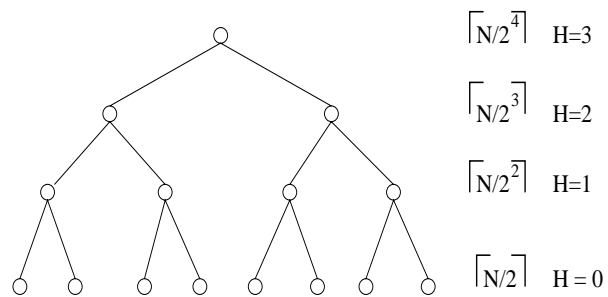
end-BUILDHEAP

```

Slide Lecture 1 -6

Number of swaps performed by BUILD-HEAP ?
naive heapify $O(N)$ \rightarrow BUILD-HEAP $O(N \log(N))$
but most heaps are small !!
IE: heapify runs faster at the lower nodes. (NOT A TIGHT BOUND)

Slide Lecture 1 -7



Slide Lecture 1 -8

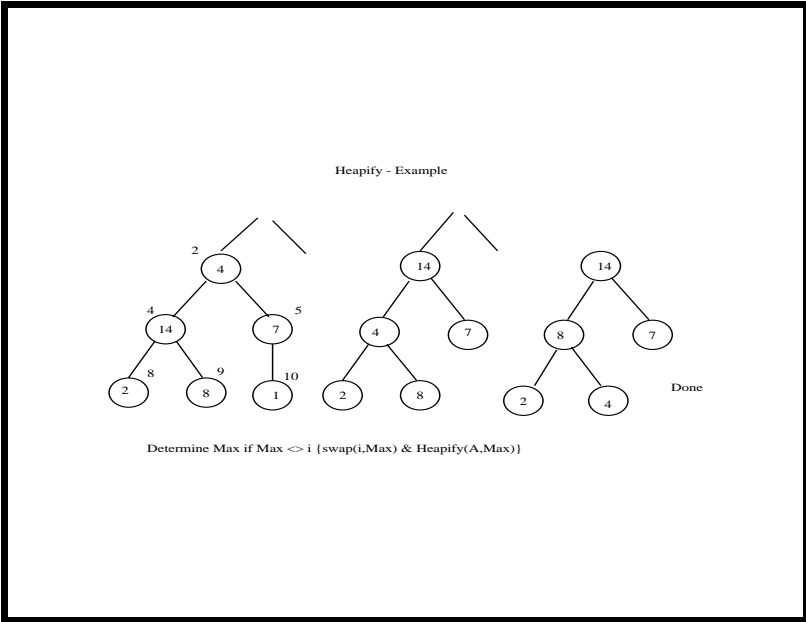
$$\sum_{H=0}^{\lfloor \lg N \rfloor} \frac{N}{2^{(H+1)}} * H$$

$$N \sum_{H=0}^{\lfloor \lg N \rfloor} H * \frac{1}{2^{(H+1)}}$$

$$\sum_{H=0}^{\infty} H * \frac{1}{2^{(H+1)}} = \frac{1/2}{(1 - 1/2)^2} = 2$$

$$N \sum_{H=0}^{\lfloor \lg N \rfloor} H * \frac{1}{2^{(H+1)}} = \mathcal{O}(N * \frac{1/2}{(1 - 1/2)^2}) = \mathcal{O}(2N) = \mathcal{O}(N)$$

Slide Lecture 1 -9



Slide Lecture 1 -10

HEAPSORT(A)

BUILDHEAP(Array)

 for i from n down to 2

 {swap(1,i);

 n:=n-1;

 Heapify(A,1);} }

COMPLEXITY:

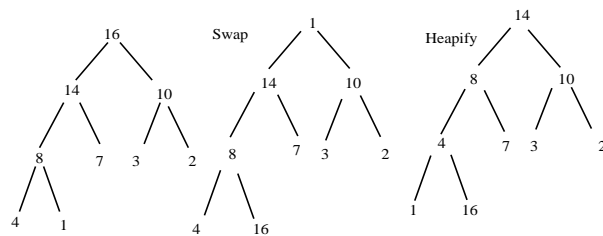
Buildheap: $O(N)$

loop : $O(N \lg N)$

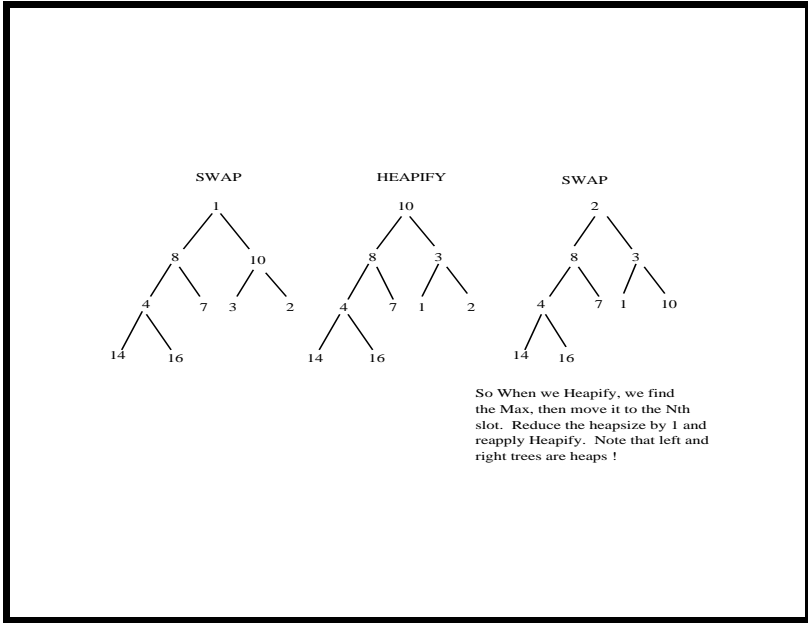
space : N (Better than MERGE SORT)

Slide Lecture 1 -11

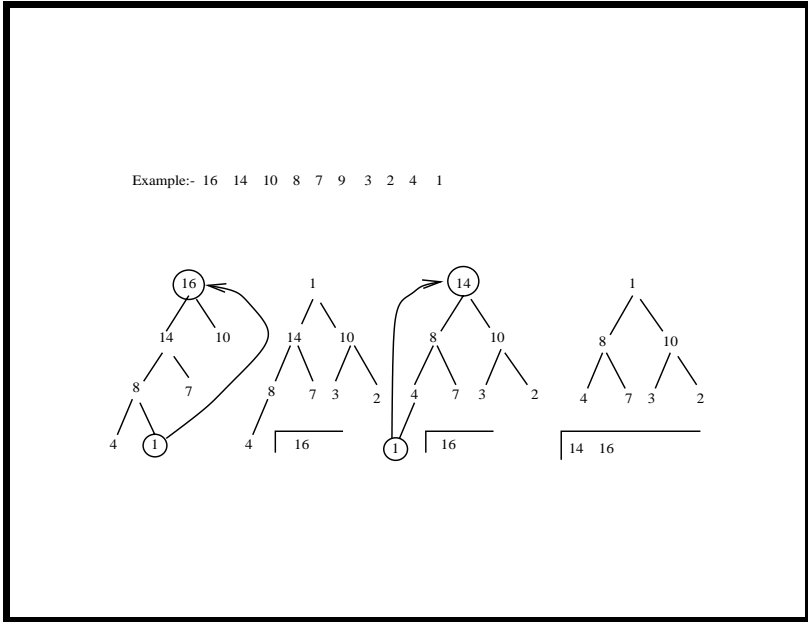
EXAMPLE



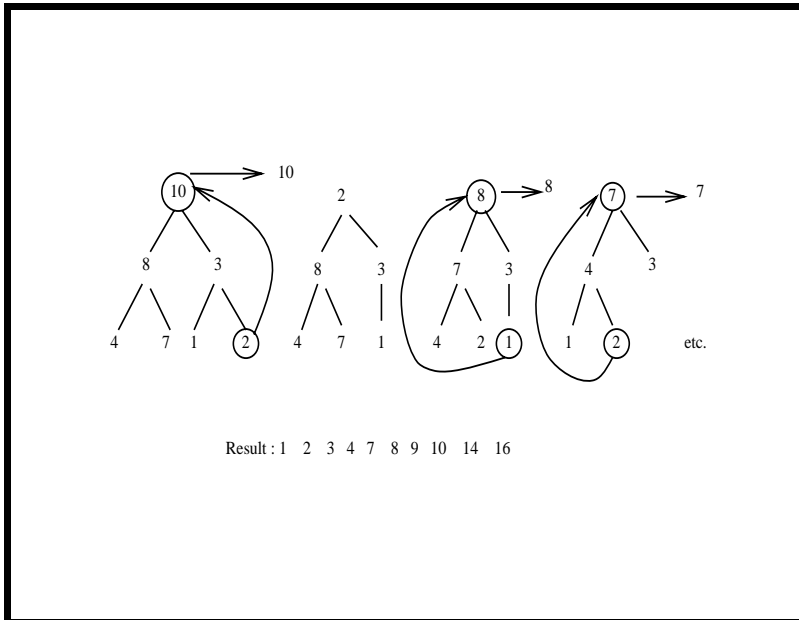
Slide Lecture 1 -12



Slide Lecture 1 -13



Slide Lecture 1 -14



Slide Lecture 1 -15

```

QUICKSORT(A,p,r)
1. if p < r
2.   then q := PARTITION(A,p,r)
3.     QUICKSORT(A,p,q)
4.     QUICKSORT(A,q+1,r)

```

To sort an entire array A , the initial call is $\text{QUICKSORT}(A,1,\text{length}[A])$.

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p\dots r]$ in place.

Slide Lecture 1 -16

```
PARTITION(A,p,r)
1. x := A[p]
2. i := p-1
3. j := r+1
4. WHILE TRUE
5.     DO REPEAT j := j-1
6.         until A[j] <= x  (less than or =)
7.         REPEAT i := i+1
8.             until A[i] >= x
9.         IF i<j then exchange(A[i], A[j])
10.        else return j

p - first position
r - last position
split using X = A[p]
```

Slide Lecture 1 -17

NOTE: We can also talk about sorting a list into values above X and below X.

Slide Lecture 1 -18

QUICKSORT WORST CASE

n-1 element in 1 list.
 0 in the other list
 1 as "dividing" element

$$T(n) = T(n-1) + n$$

$$T(n) = n + (T(n-2) + (n-1))$$

$$T(n) = n + (n-1) + (T(n-3) + (n-2))$$

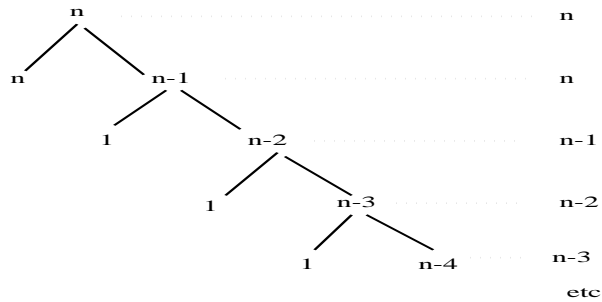
$$=$$

$$\sum_{k=1}^n K \approx \sum_{k=1}^n O(K)$$

$$= O(\sum_{k=1}^n K)$$

$$= O(n^2)$$

Slide Lecture 1 -19



NOTE there are two degenerate cases :

- j=1 sort a partition of 1 T(1)
- j=n-1 sort a partition of N T

Slide Lecture 1 -20

Instead of always picking the first element of a list as the partition element, we pick the partition element randomly. The case of picking the partition elements is extremely small.

in sorted order

Average case therefore approaches best case behavior.

We have, *BEST CASE*

$$T(n) = 2T(n/2) + O(n)$$

case 2 Master Method $O(n \log n)$

Slide Lecture 1 –21

AVERAGE

Assumes a uniform distribution of numbers to be sorted.

For randomized Quick sort (No particular input cause worst case behavior) pick the partition element at random.

Each case occurs with $1/n$ chance. Thus:

$$T(n) = (n + 1) + 1/n \sum_{q=1}^{n-1} (T(q) + T(n - q))$$

($N+1$ is the cost of partitioning: one “op” to pick random divide element and N steps to partition.)

Slide Lecture 1 –22

1	N-1
2	N-2
3	N-3
.	.
.	.
.	.
N-2	2
N-1	1

Note we never have 0, N because if partition element is smallest, or largest, we can detect this and reduce the problem, to the N-1 case.

Slide Lecture 1 -23

The change of getting a split
 $T(N) = T(K) + T(N-K) + O(n)$ or
 $T(N) = T(N-K) + T(K) + O(n)$ are equal. So:

$$T(n) = (n + 1) + 1/n \sum_{q=1}^{n-1} 2(T(q))$$

Note change of index has no effect
 Assume $T(0) = T(1) = 0$.

$$T(n) = (n + 1) + 2/n \sum_{q=1}^{n-1} (T(q))$$

Slide Lecture 1 -24

$$(1) \quad T(n) = (n + 1) + 2/n \sum_{k=1}^{n-1} T(k)$$

Multiply both sides by n.

$$(2) \quad nT(n) = n(n + 1) + 2 \sum_{k=1}^{n-1} T(k)$$

Iterate and substitute n-1 for n.

$$(3) \quad (n - 1)T(n - 1) = n(n - 1) + 2 \sum_{k=1}^{n-2} T(k)$$

Slide Lecture 1 -25

Now subtract (2) - (3)

$$nT(n) - (n - 1)T(n - 1) = 2n + 2T(n - 1)$$

$$nT(n) = 2n + 2T(n - 1) + (n - 1)T(n - 1)$$

$$nT(n) = 2n + (n + 1)T(n - 1)$$

Now a nonlinear recurrence relation

$$T(n) = 2 + \frac{(n + 1)}{n}T(n - 1)$$

Slide Lecture 1 -26

Iterate and substitute n-1 for n.

$$T(n-1) = 2 + \frac{(n)}{n-1}T(n-2)$$

$$T(n-2) = 2 + \frac{(n-1)}{n-2}T(n-3)$$

$$T(n-3) = 2 + \frac{(n-2)}{n-3}T(n-4)$$

$$T(n-4) = 2 + \frac{(n-3)}{n-4}T(n-5)$$

Hence

$$T(n) = 2 + 2\frac{(n+1)}{n} + 2\frac{(n+1)(n)}{n(n-1)} + 2\frac{(n+1)(n)(n-1)}{n(n-1)(n-2)} + \dots$$

$$T(n) = 2 + 2(n+1)\frac{1}{n} + 2(n+1)\frac{1}{(n-1)} + 2(n+1)\frac{1}{(n-2)} + \dots$$

Slide Lecture 1 -27

Note, for (n-1) term we have n,

for (n-2) term we have n-1)

But we will assume we count from n down to 1.

$$T(n) = 2 + \sum_{k=1}^n 2(n+1)/k$$

$$T(n) = 2(n+1) \sum_{k=1}^n 1/k$$

NOTE: See Cormen Pg 44, The Harmonic Series

$$\sum_{k=1}^n 1/k = \ln n + O(1) = O(\lg N)$$

Slide Lecture 1 -28