Outline

- **Divide and conquer (work trees)**
- Heap-sort
- Quick-sort

Last time … Merge-sort

Merge-sort(S={s_1, s_2, ... , s_n})
If n=1 return(S)
Else
S_1 = Merge-sort(s_1, ... , $s_{\lfloor n/2 \rfloor}$)
S_2 = Merge-sort($s_{\lfloor n/2 \rfloor +1}$, ... , s_n)
Return Merge(S_1 , S_2)

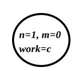
Divide and Conquer

- "Divide and conquer" is an algorithmic technique:
  - Break the problems into ***a*** sub-problems of size ***n/b***
  - Solve the sub-problems
  - Combine the solutions to the sub-problems to create a solution for the original problem
- "Divide and conquer" recurrence relations
  - ***T(1)=c=f(1)***
  - ***T(n)=a T(n/b) + f(n)***

Work Tree for Divide and Conquer

First consider the case $n=b^m$

m=0



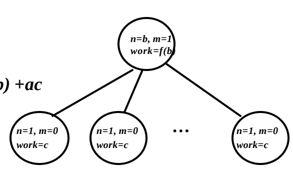
Total work: c

Work Tree for Divide and Conquer

First consider the case $n=b^m$

m=1

Total work: ***f(b) +ac***



Work Tree for Divide and Conquer

First consider the case $n=b^m$

A root with ***a*** sub-trees
- Root
  Input Size: $n=b^m$
  Work: ***f(n)***
- Each child
  Roots a work tree with Input Size b^{m-1}

Work Tree for Divide and Conquer

Properties of nodes at level i (root is at level 0):

- Input size: n/b^i
- Work: $f(n/b^i)$

Properties of level i:

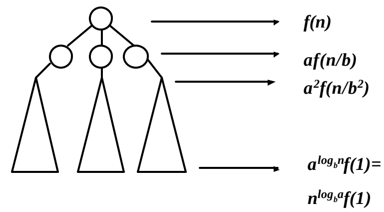
- Number of nodes at level i: a
- Total work of nodes at level i: $af(n/b^i)$

Property of tree:

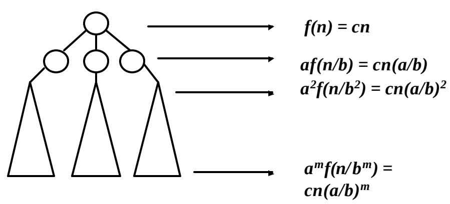
- Number of levels: $m+1$
- Total work: $\Sigma_{i=0,\ldots m} a^i f(n/b^i)$ =?

Where is "most" of the work?

f(n) is slow-growing $\leftrightarrow$ f(n) is fast-growing



$f(n)=cn,\ m=\log_b n$



$T(n) = cn\,\Sigma_{i=0\ldots m}\,(a/b)^i$

Total Work

$T(n) = cn\,\Sigma_{i=0\ldots m}\,(a/b)^i$

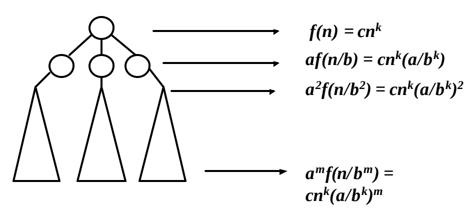
- $a<b$:
- $a=b$:
- $a>b$:

Total Work

$T(n) = cn\,\Sigma_{i=0\ldots m}\,(a/b)^i$

- $a<b$: $O(n)$
- $a=b$: $O(n\lg(n))$
- $a>b$: $O(n^{\log_b a})$

$f(n)=cn^k$



$T(n) = cn^k\,\Sigma_{i=0\ldots m}\,(a/b^k)^i$

Total Work

$T(n) = cn^k \Sigma_{i=0\ldots m} (a/b^k)^i$

$a<b^k$:

$a=b^k$:

$a>b^k$:

Total Work

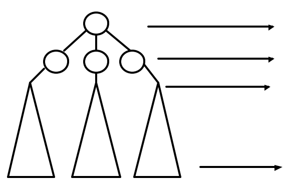
$T(n) = cn^k \Sigma_{i=0\ldots m} (a/b^k)^i$

$a<b^k$: $\theta(n^k)$

$a=b^k$: $\theta(n^k lg(n))$

$a>b^k$: $\theta(n^{\log_b a})$

$f(n)=cn^k$



$f(n) = cn^k$

$af(n/b) = cn^k(a/b^k)$

$a^2f(n/b^2) = cn^k(a/b^k)^2$

$a^mf(n/b^m) = cn^k(a/b^k)^m$

$T(n) = cn^k \Sigma_{i=0\ldots m} (a/b^k)^i$

Total Work

$T(n) = cn\, \Sigma_{i=0\ldots m} (a/b)^i$

$a<b$: $\Theta(n)$

$a=b$: $\Theta(n lg(n))$

$a>b$: $\Theta(n^{\log_b a})$

Outline

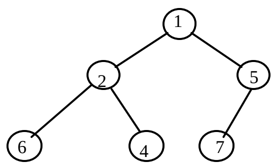
- Divide and conquer (work trees)
- **Heap-sort**
- Quick-sort

Heaps

A heap is a data-structure for storing integer that supports:

1. Build-heap(S): Return a heap on the integers in the set S.
2. Insert(x,H): Insert the integer x into the heap H.
3. Find-min(H): Return the smallest integer in the heap H.
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

Heap: {7,1,5,4,2,6}



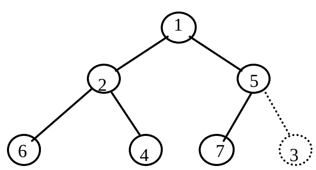
1. Rooted, binary tree, filled level by level from the left.
2. Heap property: the integer stored at a node is no larger than those of its descendents.

Heap

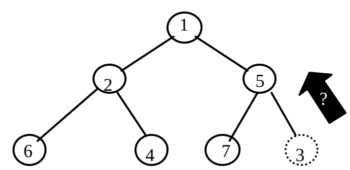
A heap is a data-structure for storing integer that supports:

1. Build-heap(S): Return a heap on the integers in the set S.
2. Insert(x,H): Insert the integer x into the heap H.
3. Find-min(H): Return the smallest integer in the heap H. O(1)
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

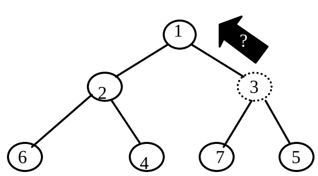
Insert(3,H) – Step 1 (add)



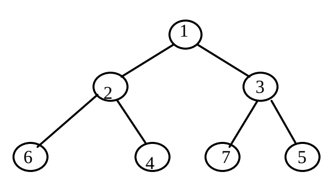
Insert(3,H) – Step 2 (bubble up)



Insert(3,H) – Step 2 (bubble up)



Insert(3,H) – return

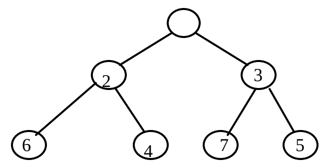


Heap

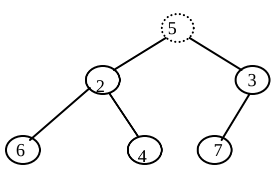
A heap is a data-structure for storing integer that supports:

1. Build-heap(S): Return a heap on the integers in the set S.
2. Insert(x,H): Insert the integer x into the heap H. O(lg n)
3. Find-min(H): Return the smallest integer in the heap H. O(1)
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

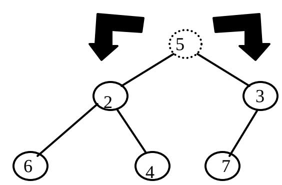
Extract-min(H) – Step 1 (remove)



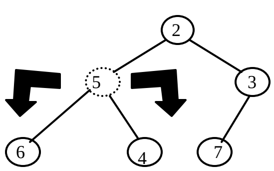
Extract-min(H) – Step 2 (move last to root)



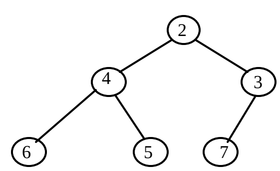
Extract-min(H) – Step 3 (bubble down)



Extract-min(H) – Step 3 (bubble down)



Extract-min(H) – return



Heap

A heap is a data-structure for storing integer that supports:

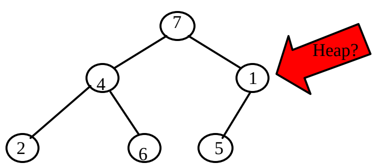
1. Build-heap(S): Return a heap on the integers in the set S.
2. O(lg n) Insert(x,H): Insert the integer x into the heap H.
3. O(1) Find-min(H): Return the smallest integer in the heap H.
4. O(lg n) Extract-min(H): Remove the smallest integer from the heap H and return it.

Heap

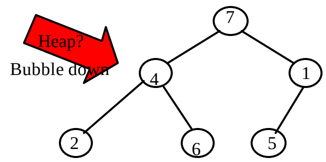
A heap is a data-structure for storing integer that
O(n) supports:

1. O(lg n)) Build-heap(S): Return a heap on the integers in the set S.
2. O(lg n) Insert(x,H): Insert the integer x into the heap H.
3. O(1) Find-min(H): Return the smallest integer in the heap H.
4. O(lg n) Extract-min(H): Remove the smallest integer from the heap H and return it.

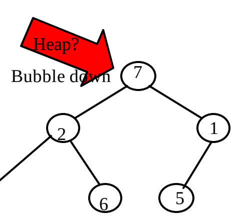
Build-heap{7,1,5,4,2,6}



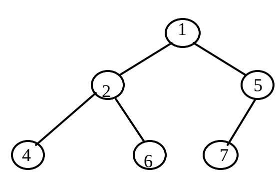
Build-heap{7,1,5,4,2,6}



Build-heap{7,1,5,4,2,6}



Build-heap{7,1,5,4,2,6}



Running Time

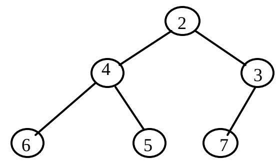
Level 0: 1 node, takes clg(n)
Level 1: 2 nodes, each takes c·lg(n/2)

Level i: $2^{i\ \text{nodes}}$, each takes c·lg($n/2^i$)

Level lg(n/2): n/2 nodes, each takes c

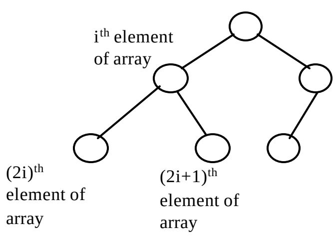
Time: $\Sigma_{i=0\ldots lg(n/2)} 2^i\, c lg(n/2^i) = O(n)$

Implementing a heap in an array



2	4	3	6	5	7

Array Indexing



Heap-sort(S)

H=Build-heap(S) ⟶ O(n)
For i=1 to n
  S(i) = Extract-min(H) ⟶ O(lg n)
Return

Heap-sort is O(n lg n)

Outline

- Divide and conquer (work trees)
- Heap-sort
- **Quick-sort**

Quick-sort(S)

- If ||S||≤1 then return
- Choose a pivot s from S
- S1={t ε S-{s} | t > s}
- S2={t ε S-{s} | t ≤ s}
- Return Quick-sort(S1),s,Quick-sort(S2)

Quick-sort(3,1,5,2,4)

Pivot rule: Choose first element in list

Quick-sort(3,1,5,2,4)
Quick-sort(1,2), 3, Quick-sort(5,4)
Quick-sort(), 1 , Quick-sort(2), 3, Quick-sort(5,4)
1, 2, 3, Quick-sort(5,4)
1,2,3, Quick-sort(4), 5, Quick-sort()
1,2,3,4,5

Analysis

- Quick-sort is correct: Inductive argument
- Quick-sort is $O(n^2)$
- Average case analysis of quick-sort.

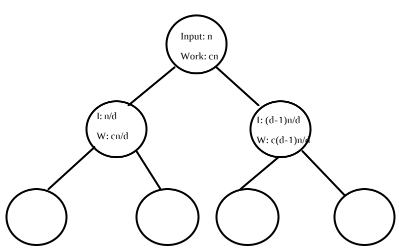
Average-case analysis

What does average-case mean?
- Deterministic algorithm with a known input distribution
- Randomized algorithm on any (i.e. worst-case) input

Suppose …

- We have a pivot rule such that for some $d>1$
  - n/d elements are no less than the pivot
  - n/d elements are no greater than the pivot
- $T(n) \le T(n/d) + T((d-1)n/d) + c$

Work Tree



Running Time

- Work done at level i: $O(cn)$
- Number of levels: $O(\log_{(d-1)/d} n)$
- Running time: $O(n \lg(n))$

Randomized Quick-sort(S)

- If $||S|| \leq 1$ then return
- Choose a pivot s uniformly at random from S
- $S1=\{t \in S-\{s\} \mid t > s\}$
- $S2=\{t \in S-\{s\} \mid t \leq s\}$
- Return Quick-sort(S1),s,Quick-sort(S2)

Average-case analysis

What does average-case mean?

- Deterministic algorithm with a known input distribution
- **Randomized algorithm on any (i.e. worst-case) input**

A brief tour of (discrete) probability theory…

- Sample space and elementary events
- Discrete probability distributions
- Discrete random variables
- Expectation

The experiment

- A fair coin is flipped
- Sample space: {Head, Tail}

The experiment

- Two fair coins are flipped
- Sample space: {HH, HT, TH, TT}

Discrete Probability Distribution

Assigns a real number to outcomes:

- Experiment 1: P(H)=P(T)=1/2
- Experiment 2:

  P(HH)=P(HT)=P(TH)=P(TT)=1/4

Discrete Probability Distribution Properties

- $P(A) \geq 0$
- $P(S) = 1$
- $P(A \cup B)=P(A)+P(B)$ when A and B are disjoint events.

9/11/00 CS140(4) 55

Discrete Random Variable X

- Sample space is a finite set of real numbers.
- X is the outcome of the experiment
- Probability distribution: $P(X=x)$
- Expected value of X:

  $E[X] = \sum_{x \varepsilon S} x\, P(X=x)$

  $E[X^2]= \sum_{x \varepsilon S} x^2\, P(X=x)$

  $Var(X) = E[(X-E[X])^2]$

9/11/00 CS140(4) 56

Example

- A unfair coin is tossed n times:

  $P(H)=p$, $P(T)=1-p=q$
- X is the number of heads
- $P(X=k) = C(n,k)p^k q^{n-k}$

  (Binomial distribution)
- $E[X] = np$
- $E[X^2] = npq + n^2p^2$

9/11/00 CS140(4) 57