

Computer Science Lunch

- ◉ WHEN: 12:15pm, Thursday, Feb. 2
- ◉ WHERE: Kendade 307
- ◉ TOPIC: Sarah Tulimat will talk about her study away experience in Budapest

1

Big O Notation (review)

- ◉ Let $T(n)$ be a function that defines the worst-case running time of an algorithm.
- ◉ $T(n)$ is $O(f(n))$ if
 $T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$
- ◉ Example:
Let $T(n) = 3n + 2$
 $T(n)$ is $O(n)$ because
 $T(n) \leq 4n$ for all $n \geq 2$
- ◉ $O(n)$ is the asymptotic upper bound of $T(n)$.

2

Transitivity

Claim: If f is $O(g)$ and g is $O(h)$, then f is $O(h)$

How would you prove that?

3

Additivity (1)

Claim: If f is $O(h)$ and g is $O(h)$, then $f+g$ is $O(h)$

How would you prove that?

4

Additivity (2)

Claim: If f is $O(g)$, then $f+g$ is $\Theta(g)$

How would you prove that?

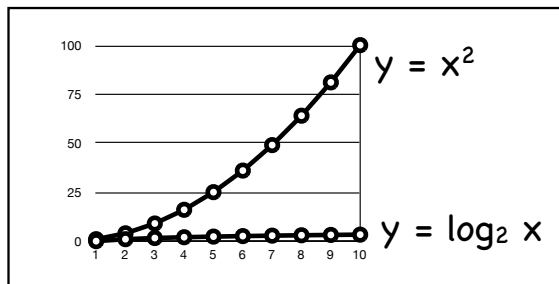
5

Other Asymptotic Orderings

Logarithms:

$\log_a n$ is $O(n^d)$, for all bases a and all degrees d

→ All logarithms grow slower than all polynomials



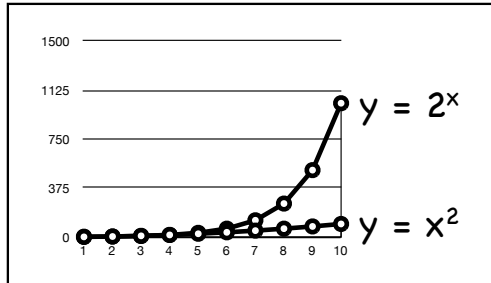
6

Other Asymptotic Orderings

- Exponential functions:

n^d is $O(r^n)$ when $r > 1$

→ Polynomials grow no more quickly than exponential functions.



7

A Harder Example

- Which of these grows faster?

$$n^{4/3}$$

$$n(\log n)^3$$

8

Linear Time - $O(n)$

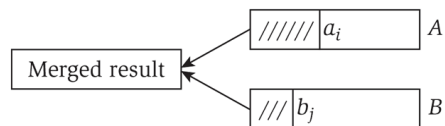
- Linear time. Running time is at most a constant factor times the size of the input.
- Computing the maximum. Compute maximum of n numbers a_1, \dots, a_n .

```
max = a1
for i = 2 to n {
  if (ai > max)
    max = ai
}
```

9

What is the algorithm?

- Merge. Combine two sorted lists $A = a_1, a_2, \dots, a_n$ with $B = b_1, b_2, \dots, b_n$ into sorted whole.



10

$O(n)$ - Why?

```
i = 1, j = 1
while (both lists are nonempty) {
  if (ai ≤ bj) {
    append ai to output list
    increment i
  }
  else {
    append bj to output list
    increment j
  }
}
append remainder of nonempty list to output list
```

11

Logarithmic time - $O(\log n)$

⦿ Logarithmic time. Do a constant amount of work to discard a constant fraction of the input (often 1/2)

⦿ Binary search: Search a sorted collection

```
low = 0;
high = a.length - 1;
while( low <= high ) {
  middle = ( low + high ) / 2;
  if( key == a[ middle ] )
    return middle;
  else if( key < a[ middle ] )
    high = middle - 1;
  else
    low = middle + 1;
}
return -1;
```

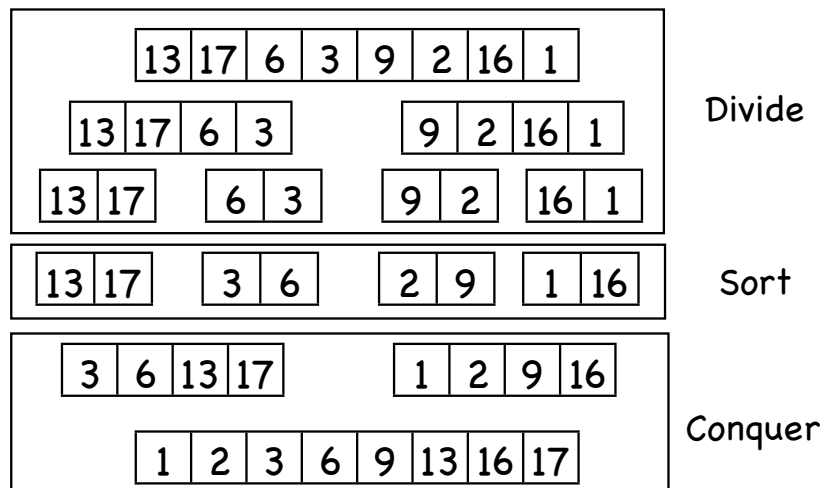
12

Linearithmic time - $O(n \log n)$

- Linearithmic time. Divide-and-conquer
 - Repeatedly divide a problem in half (gives the $\log n$ part)
 - Solve the problem in constant time
 - Combine the results of the divided problems in linear time
- Mergesort:
 - Repeatedly divide collection in half
 - Sort when collections have size 2
 - Merge resulting lists

13

Mergesort



14

Quadratic Time - $O(n^2)$

- Quadratic time.
 - Examine all pairs of input
 - Typically involves nested loops
- Closest pair of points in a plane. Given a list of n points in the plane $(x_1, y_1), \dots, (x_n, y_n)$, find the pair that is closest.

```
min = (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
  for j = i+1 to n {
    d = (xi - xj)2 + (yi - yj)2
    if (d < min)
      min = d
  }
}
```

15

Beyond Polynomial Time

- $O(N!)$. Consider all permutations and pick the best.
- Traveling salesperson. Given n cities with the distances between each pair of cities. Find the shortest path that visits each city exactly once.

```
min = sum of distances visiting
cities in order c1, c2, ... cn
shortest_path = c1, c2, ... cn
for every other permutation p
  cost = sum of distances of
  that permutation
  if (cost < min) {
    min = cost
    shortest_path = p
  }
```

16

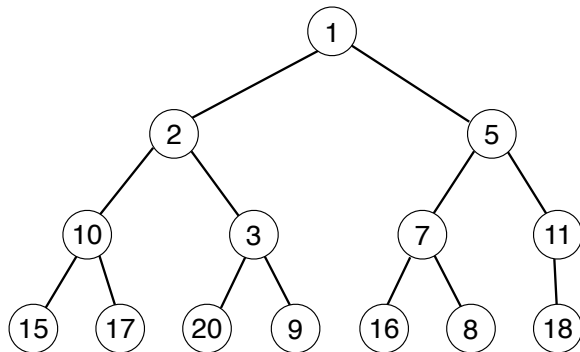
Priority Queues

- Given a collection, we want to:
 - Add an element
 - Find the element with the highest priority
 - Remove the element with the highest priority

17

Priority Queue Implementation

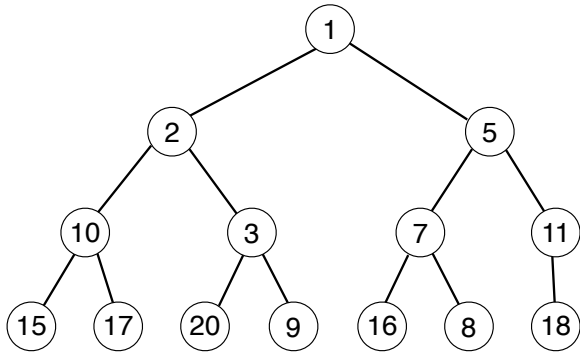
- Heap
 - Balanced binary tree, filled from left to right
 - Lower key \Rightarrow higher priority
 - Heap order: For every node n , $\text{key}(n) \geq \text{key}(\text{parent}(n))$



18

Find Smallest

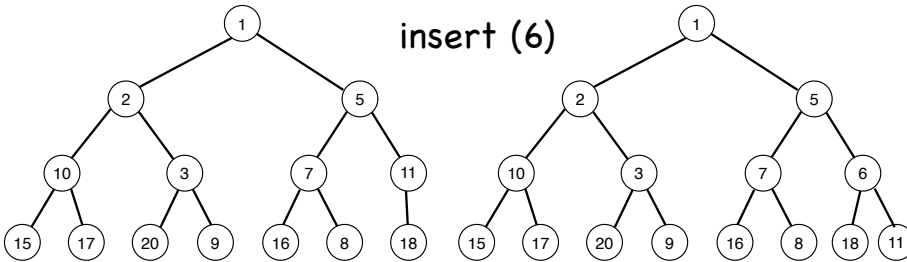
- What is the algorithm?
- What is its running time?



19

insert

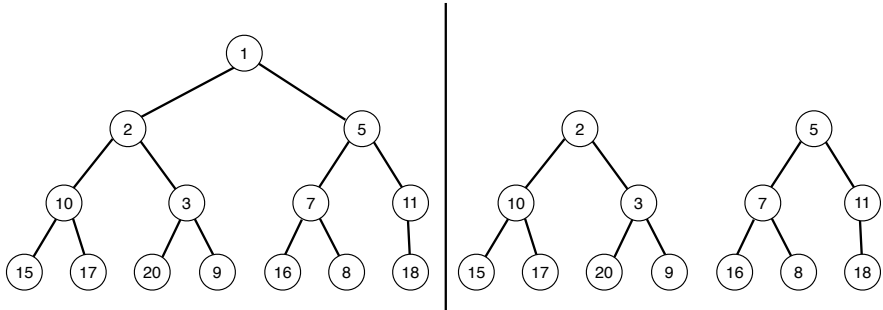
insert (6)



What is the algorithm?

20

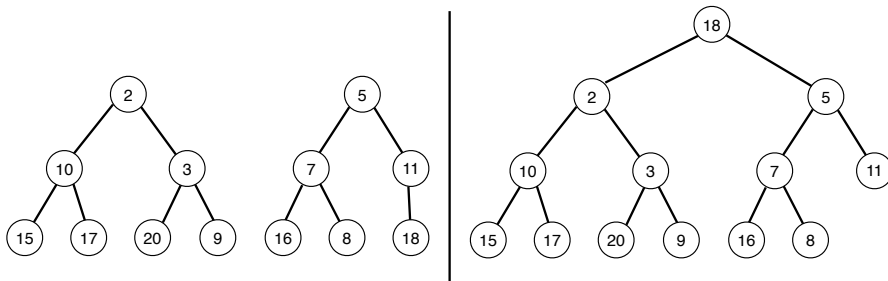
removing the smallest



- ◀ Finding the smallest and removing it is easy
- ◀ How do we merge the 2 trees back into a heap?

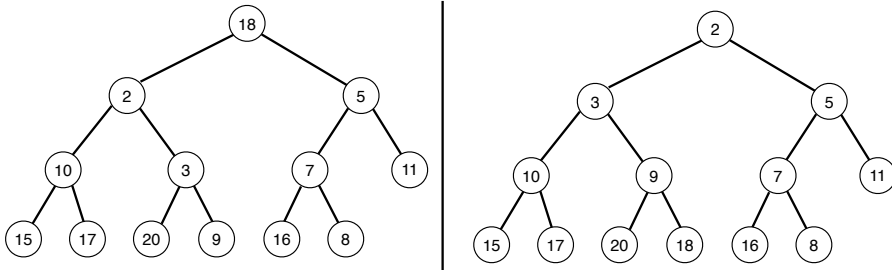
21

Step 1: Move leaf to root



22

Step 2: Swap with children to restore heap



23

Summary

- Asymptotic bounds give an implementation-independent definition of an algorithm's running time.
- It's generally safe to ignore constant factors because constants tend to be small in real algorithms.
- $O(\log n)$ is better than $O(n^d)$ which is better than $O(r^n)$

24

Summary

- $O(\log n)$ - generally dividing problem in 1/2 on each iteration
- $O(n)$ - operate on each input value
- $O(n \log n)$ - divide-and-conquer
- $O(n^2)$ - operate on each pair of inputs
- $O(n!)$ - operate on each permutation of inputs