upper bound = $O(n)$
average case = $\theta(n)$
lower bound = $\Omega(n)$

Kamryn Parker

CS 321 Data Structures (Spring 2020)

Homework #1 (80 points), Due Date: 2/18/2020 (Tuesday)

- **Q1(12 points): Asymptotic Notations**

anything that is tight bound of n then you know lower bound of n

$n - + n\downarrow = n\uparrow$ (a)(4 points) Which one of the following is a wrong statement?

1. $\Theta(n) + O(n) = \Omega(n)$
[✓ 2.] $O(n) + \Omega(n) = \Theta(n)$ $n \downarrow + n \uparrow = n$ —— so can't find tight bound of n
↖ mon significant
3. $\Theta(n) + \Omega(n) = \Omega(n)$ — $n - + n\uparrow = n\uparrow$
4. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$ from θ and Ω

(b)(4 points) Which one of the following sorting algorithms will have the best best-case running time?

1. Selection sort → $\theta(n^2)$
[✓ 2.] Insertion sort → $\theta(n)$ - LINEAR!
3. Heap sort → $\theta(n \log_2 n)$
4. Quick sort → $\theta(n \log_2 n)$

(c)(4 points) Explain why the statement, "The running time of an algorithm is $\Omega(1)$," is meaningless.

It is irrelevant that the the lower bound is a constant because as time increases the constant value is disregarded. Also it can be a given that the lowest bound would be a constant value because that is the lowest a bound could ever go. This is also saying that $\Omega(1)$ is constant

* any time something is changing by a factor is a log

- **Q2(18 points): Running Time and Growth of Functions**

(a)(10 points) Assume evaluating a function $f(n)$ in the pseudocode below takes $\Theta(n)$ time.

```
i = 1;
sum = 0;
while (i <= n)   log₂n
      do if (f(i) > k)
            then sum += f(i); θ(1) + θ(2) + θ(4) + θ(8) + ... + ~θ(n)     2^k
         i = 2*i;  -> constant
```

What is the running time (use an asymptotic notation) of the above code? Justify your answer. The if statement would happen every time.

The while loop executes every time and doubles i each time. The function grows at a linear rate. The number of terms is the number of runs.

The final run would be $2^{\log_2 n}$

$$\sum_{k=0}^{n} x^2 = 1 + x + x^2 + \dots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

$$\theta(1+2+4 \dots + n)$$

$$\theta(1+2^1+2^2 \dots + 2^{\log_2 n})$$

$$\theta\left(\sum_{k=0}^{\log_2 n} 2^k\right) = \frac{2^{\log_2 n+1} - 1}{2-1}$$

$$2^x = n$$

$$\log_2 2^x = \log_2 n$$

$$x = \log_2 n$$

$$= \theta\left(2^{(\log_2 n+1)} - 1\right) = \theta\left(2^{\log_2 n+1}\right) = \theta\left(2 \cdot 2^{\log_2 n}\right) = \theta(2 \cdot n) = \underline{\theta(n)}$$

geometric equation, AS in the book

(b)(8 points) For the following functions, please list them again but in the order of their asymptotic growth rates, from the least to the greatest. For those functions with the same asymptotic growth rate, please underline them together to indicate that.

$$\cancel{3^n},\ (\cancel{\log_2} n)^n,\ \cancel{\log_2} n^n,\ \cancel{n}^2,\ \cancel{\log_2} n^{10},\ \cancel{\sqrt{n}},\ 5^{\cancel{n}/2}, \log_{\cancel{10}}(n!)$$

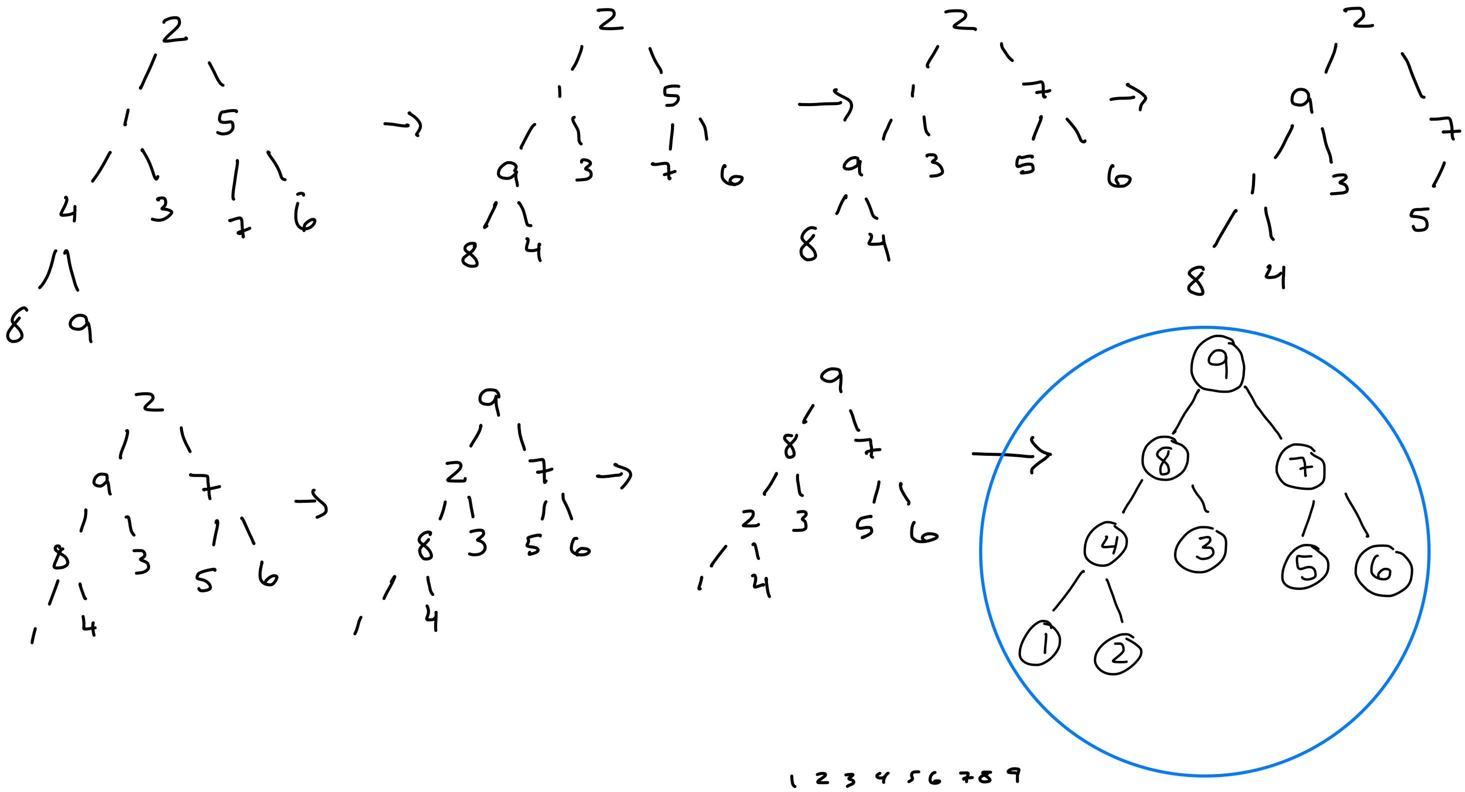
$$\underline{\log_2 n^{10}, \log_2 n^2}, \sqrt{n}, \log_{10}(n!), n^2, 5^{n/2}, 3^n, (\log_2 n)^n$$

$$5^{n/2} = \sqrt{5}^n$$

$\log n! = n \log n$ ← approximate

- **Q3(28 points): Sorting**

(a)(7 points) For a given input array $A: <2, 1, 5, 4, 3, 7, 6, 8, 9>$, what is the sequence of numbers in A after calling Build-Max-Heap(A) ? (please show the intermediate trees).



1 2 3 4 5 6 7 8 9

(b)(7 points) For a given input array $A: <4, 2, 9, 1, 8, 6, 7, 3, 5>$, what is the sequence of numbers in A after the first partition (by calling `Partition(A, 1, 9)`)? Note that 1 and 9 in `Partition(A, 1, 9)` function call are array indexes.

pivot or A[r] = 5

first element j = p   p=0

i = p-1

1 2 3 4 5 6 7 8 9

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑ ↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑ ↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑ ↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑ ↑

i j

A: < 4, 2, 9, 1, 8, 6, 7, 3, 5 >

↑ ↑

i j ← swap

A: < 4, 2, 1, 9, 8, 6, 7, 3, 5 >

↑ ↑

i j

A: < 4, 2, 1, 9, 8, 6, 7, 3, 5 >

↑ ↑

i j

(c)(8 points) By using the Max-Heap data structure to implement a priority queue, some applications may need to change the data (priority) of a specific node i. That is, given an index i, change the priority of node i to a new priority t. Please write a pseudocode for this procedure. You can implement the procedure by calling the `MaxHeapifyDown(A, i)` and/or `MaxHeapifyUp(A, i)` methods.

```
Max-Heap-update(A, i, t)
{
    p1 ← A[i];

    if p1 < t {
        p1 ← - - t;
        MaxHeapify Up ( i );
    } else {
        p1 ← - - - t;
        Max Heapy Down (i);
    }

}
```

(d)(6 points) Please describe how to use a priority queue to implement a queue abstract.

You would allow the priority queue to enqueue and dequeue I would have the inputs be the same priority and determine how they are ordered based on their arrival time because that is how priority queue would break a tie on priorities normally. This way nothing is changed about the priority queue. So when I call dequeue it would remove the oldest time in the list and enqueue would be the newest time

- **Q4(22 points): Linear Time Sorting**

(a)(6 points) Please describe the reason(s) why we choose the `counting sort` algorithm to sort each digit in the `Radix Sort`?

Counting sort is a sorting algorithm that assumes each number of n elements is an integer in the range 0 to k. So we can assume $k = O(n)$ Counting sort is also a stable sorting algorithm so it can break ties between values by knowing which value appears first in the input then appears first in the output.

We use this algorithm for Radix sort because radix is not a stable algorithm so counting sort is useful because of its stability and if we can assume that for the running time of Radix $d * \Theta(n+k) = \Theta(dn + dk)$ with counting that if $k = O(n)$ like counting sort then we would have a linear run time $\Theta(n)$!

$\log_2$ is 1 or 0

(b)(6 points) What is the best running time to sort n integers in the range $[0, n^2]$, and How?

radix base 10

You would use a non-comparison sort because those are linear sorting algorithms which give us a linear time.

$$\log_r n^2 = d \qquad \Theta(d \cdot (n+k)) = \Theta(dn + dk)$$

k-1

$$\log_n n^2 = d \rightarrow \log_n n^2 = 2 \rightarrow \log_n n^d = d + 1$$

You would express the values of radix n so that as a $\log_n n^2 = d$ then you would be able to find a constant as d. The noncomparison sort I would use would be radix sort with counting sort so I will get $\Theta(d \cdot (n+k)) = \Theta(dn + dk)$ which then can be solved by have $\log_n n^d = d+1$ $k = n-1$ and therefore k is $O(n)$. So by showing that K is an $O(n)$ runtinae and d is a constant you therefor can confimm that $\Theta(dn + dk)$ is $= \Theta(n)$

(c)(10 points) Given an input array A with n integers in $[0, k]$, we can use the array C in the counting sort to find out how many integers in A are in a range $[a, b]$. Write a pseudocode for this query. Assume $C[i]$ already contains the number of input integers $\leq i$.

```
FindNumIn(a, b) // both a and b are integers

  C[i]
  C[b];
  C[a];

  if (a > k or b > k) {
      throw exception "values ar more than input"
  } else if (a > b) {
      throw exception "input a larger than input b"
  } else if (a < 0) {
      throw exception "input a is invali'd"
  } else {

      return C[b] - C[a-1]
  }
```