**CSCI 3104: Algorithms**
**Spring 2022**
**Recitation #8 - Divide & Conquer**

**Midterm TA reviews.** Please take take a few minutes to fill out TA FCQs! They're very helpful for us TAs. You can find the link in your email under the subject line "Computer Science Midterm TA FCQ's."

## Problem 1

Say we have an algorithm which divides the input of size $n$ into three sets of sizes $n/2$, $3n/8$, and $n/8$, with base cases when $n \leq 8$.

a. Write down the recurrence for the runtime of the algorithm.

b. Draw the first few levels of the recursion tree, labelling each vertex with its input size.

c. Is this a balanced or unbalanced recursion?

d. Give an upper bound on the runtime for our algorithm, as tight as you can.

## Problem 2

In merge-sort, we sort a list by dividing it into two halves, sorting the halves recursively, then merging them together. What's so special about breaking things into two lists? Why not three? you could think about what we'll call 3-Mergesort, where you split the list into three lists of size roughly $n/3$, and sort the three sub-lists recursively, then merge them. 3-Mergesort works as follows, assuming $n$ is always nicely divisible (to leave out floors):

**Algorithm 1** 3-Mergesort

```
1: procedure 3-MERGESORT(Integer array $L$)
2:     $n \leftarrow |L|$
3:     $A \leftarrow$ 3-MERGESORT($L[0 : n/3 - 1]$)
4:     $B \leftarrow$ 3-MERGESORT($L[n/3 : 2n/3 - 1]$)
5:     $B \leftarrow$ 3-MERGESORT($L[2n/3 : n]$)
6: return 3-MERGE($A$, $B$, $C$)
```

a. Write pseudocode for 3-Merge. It should take as input three sorted lists and output them as a single sorted list containing the elements of all three lists.

b. Give the recurrence relation of 3-Mergesort, and give an asymptotic bound. Is our new sorting algorithm better than regular Mergesort?

c. Is the recurrence tree balanced or unbalanced?

d. How might we implement (deterministic) 3-Quicksort? Do you think this be an improvement?

   (You don't need to give a formal statement of the algorithm, just give a description.)

## Problem 1

Say we have an algorithm which divides the input of size $n$ into three sets of sizes $n/2$, $3n/8$, and $n/8$, with base cases when $n \leq 8$.

a. Write down the recurrence for the runtime of the algorithm.

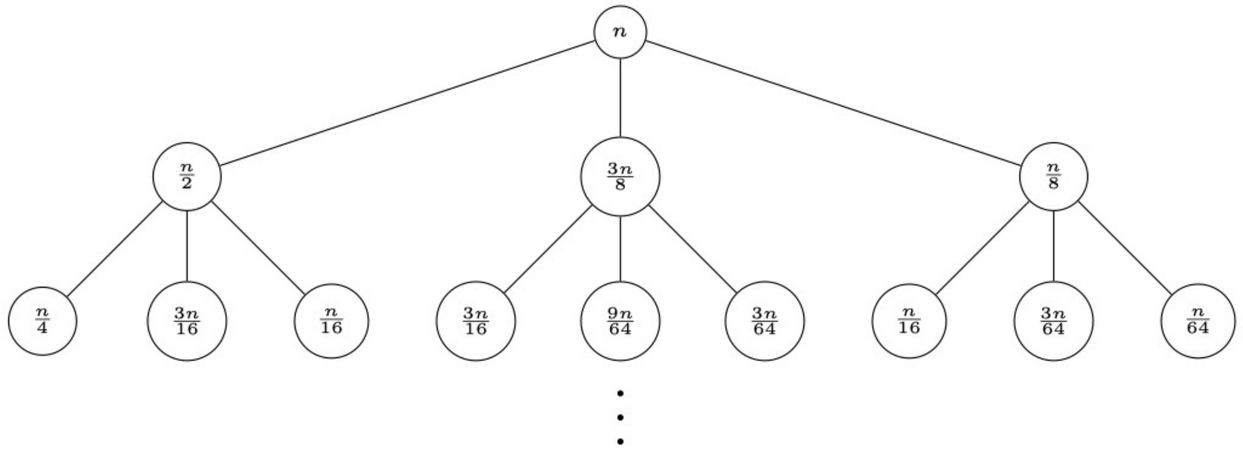
$$T(n) = \begin{cases} \Theta(1), & n \leq 8 \\ T(\frac{n}{2}) + T(\frac{3}{8}n) + T(\frac{n}{8}) + \Theta(n), & n > 8 \end{cases}$$

b. Draw the first few levels of the recursion tree, labelling each vertex with its input size.

c. Is this a balanced or unbalanced recursion?

d. Give an upper bound on the runtime for our algorithm, as tight as you can.

b.



c.

This tree is balanced, since each branch is a constant fraction of the input size.
In an unbalanced tree, we remove a fixed number of elements at each recursion, rather than a fraction.

Spits like (4, n-4) are unbalanced partitioning (i.e., one problem size is constant), and splits like (n/5, 4n/5) are balanced partitioning (i.e., constant ratio split). Split (1, n-1) is maximally unbalanced split, and (n/2, n/2) is maximally balanced split

d. Give an upper bound on the runtime for our algorithm, as tight as you can.

We compute an upper bound as follows:

(a) *Compute the amount of work done at each level.* We write out the total work at several levels, noting that the work is $O(m)$ for input of size $m$:

$$\begin{gathered} n = n \\ n/2 + 3n/8 + 5n/8 = n \\ (n/4 + 3n/16 + n/16) + (3n/16 + 9n/64 + 3n/64) + (n/16 + 3n/64 + n/64) = n \\ \vdots \end{gathered}$$

We note the pattern that each row requires total work $n$ (as long as no branches have reached a base case). Thus, we do at most $O(n)$ work in each layer.

(b) *Compute the longest path.* We upper bound the number of possible layers by the depth of the deepest root-base case path in the tree. Let $k$ denote the depth of the longest path. Since the largest recursion at each level is of size $n/2$, the path where we always take $n/2$ is the longest path, we stop when $n/(2^k) \leq 8$. Thus, the path must terminate by depth $\log_2(n) - 3$.

(c) *Upper-bound the runtime.* We upper bound the runtime by assuming that we reach no leaves (base cases) before the longest path terminates. Thus, our runtime is

$$T(n) \leq n\,(\log_2(n) - 3) \in O(n \log n)$$

(*Note:* we could probably achieve a better runtime bound, but this is good enough.)

## Problem 2

In merge-sort, we sort a list by dividing it into two halves, sorting the halves recursively, then merging them together. What's so special about breaking things into two lists? Why not three? you could think about what we'll call 3-Mergesort, where you split the list into three lists of size roughly $n/3$, and sort the three sub-lists recursively, then merge them. 3-Mergesort works as follows, assuming $n$ is always nicely divisible (to leave out floors):

**Algorithm 1** 3-Mergesort

```
1: procedure 3-MERGESORT(Integer array $L$)
2:     $n \leftarrow |L|$
3:     $A \leftarrow$ 3-MERGESORT($L[0 : n/3 - 1]$)
4:     $B \leftarrow$ 3-MERGESORT($L[n/3 : 2n/3 - 1]$)
5:     $B \leftarrow$ 3-MERGESORT($L[2n/3 : n]$)
6: return 3-MERGE($A$, $B$, $C$)
```

a. Write pseudocode for 3-MERGE. It should take as input three sorted lists and output them as a single sorted list containing the elements of all three lists.

b. Give the recurrence relation of 3-MERGESORT, and give an asymptotic bound. Is our new sorting algorithm better than regular MERGESORT?

c. Is the recurrence tree balanced or unbalanced?

d. How might we implement (deterministic) 3-QUICKSORT? Do you think this be an improvement?

(You don't need to give a formal statement of the algorithm, just give a description.)

a

```
procedure. 3-MERGE ( Integer Arrays : A, B, C )
    L = φ
    while A or B or C is not empty do.
        a ← A[0];  b ← B[0];  c ← C[0];
        x = min (a, b, c);  X = array containing x;
        Remove x from X;  Append x to L;
    return L.
```

b.

$$T(n) = 3T\left(\frac{n}{3}\right) + O(n)$$
$$= 3^2 T\left(\frac{n}{3^2}\right) + 3\,O\left(\frac{n}{3}\right) + O(n)$$
$$= 3^3 T\left(\frac{n}{3^3}\right) + 3^2 O\left(\frac{n}{3^2}\right) + 3\,O\left(\frac{n}{3}\right) + O(n)$$
$$= O(n) + \dots + O(n) = (\lceil \lg_3 n \rceil)\, O(n) = O(n \lg n).$$

$$\boxed{\frac{n}{3^k} < 3 \Rightarrow k > \lg_3 \frac{n}{3} \Rightarrow k = \lceil \lg_3 n \rceil}$$

$$\frac{1}{2} + \frac{1}{3} + \frac{3}{6}$$

c. Is the recurrence tree balanced or unbalanced?

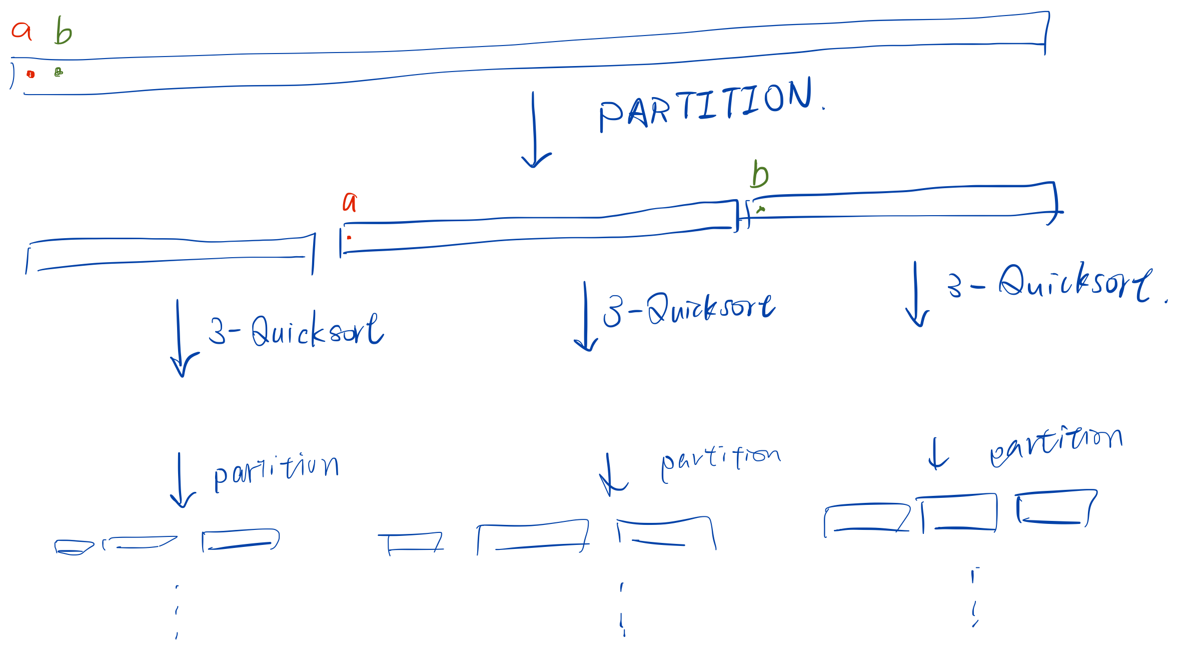
It's balance. The reason is same as. 1. (c).

d. How might we implement (deterministic) 3-QUICKSORT? Do you think this be an improvement?

(You don't need to give a formal statement of the algorithm, just give a description.)

Write a Patition function, which creates 3 segments of an input list, according to two given pivots. This takes $O(n)$, where $n$ is the length of the input,

In the 3-Quicksort, we firstly select two pivots a and b, such that $a \le b$. Then apply the partition function, given a. and b, to have 3 segments of the input list. Finally, we can then sort these 3 sections recursively.



Through the worst-case analysis, 3 Quicksort does not improve. When $a=b$, 3-Quicksort is the same as 2-Quicksort. Thus, 3-Quicksort' worse case is at least as worse as 2-Quicksort's worse case.