Submission for Problem Set 2 (due Wednesday, October 3, 11:59 PM)

Name: Deepanshu Parihar

Problem 1. (2 points) Binary Search Implementation that Passes Arrays

Consider the following implementation of the recursive binary search algorithm that passes arrays in its recursive calls.

```
BinarySearch(A, n, x):
if n = 1
  if x = A[1] return Yes
  else return No
else
  m = ⌊n/2⌋
  if x = A[m] return Yes
  if x < A[m]
    copy A[1...m] into a new array B[1...m]
    BinarySearch(B, m, x)
  else
    copy A[m+1...n] into a new array B[1...n-m]
    BinarySearch(B, n-m, x)
```

Give a recurrence for the worst-case running time of the above binary search implementation. Solve the recurrence to derive a tight bound on the running time of the above algorithm.

Answer:
Assuming that n is even and the array is sorted.

The recurrence for the worst-case running time of the above binary search implementation is $T(n) \leq n/2 + T(n/2)$.

Taking the base case T(1)=1

Now, solving the recurrence-

$T([]) \leq T([]/2) + []/2$

$T(n) \leq n/2 + T(n/2)$.

$T(n) \leq n/2 + n/4 + T(n/4)$.

$T(n) \leq n/2 + n/4 + n/8 + T(n/8)$.

So we get a series which is-

$T(n) \leq n/2 + n/4 + n/8 + n/16 + ...$

$T(n) \leq n/2(1 + 1/2 + 1/4 + 1/8 + ...)$

Sum of such geometric progression is 1 / 1-r

r here is $1/2$

So the sum is 1 / (1 - 1 /2)

$T(n) \leq (n/2) * 2$

$T(n) \leq \Theta(n)$

So we get $T(n) = \Theta(n)$

Problem 2. (2 points) A variant of Mergesort

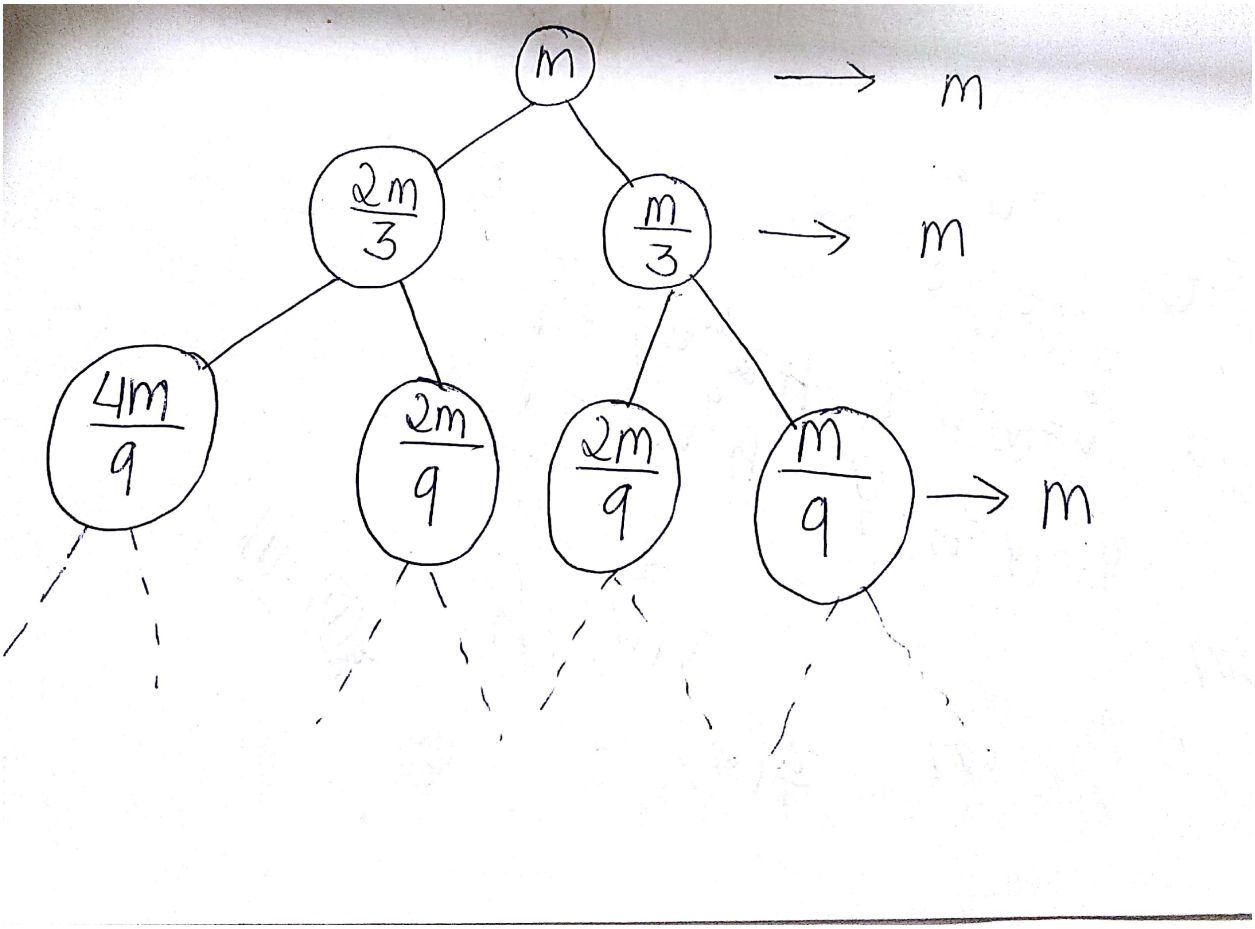
Consider the following variant of Mergesort. If the array has more than two elements, it recursively sorts the first two-third and the last one-third. Then, it merges the first two-third with the last one-third. If the array has at most two elements, then it trivially sorts the array. For completeness, here is the pseudocode for sorting the array $A[\ell \ldots r]$.

```
NEWSORT(A, ℓ, r):
if r − ℓ + 1 = 1:
  exit
if r − ℓ + 1 = 2 and A[ℓ] > A[r]:
  swap A[ℓ] ↔ A[r]
  exit
if r − ℓ + 1 > 2:
  m ← ⌊(r − ℓ + 1)/3⌋
  NEWSORT(A, ℓ, ℓ + 2m − 1)          [Sort first two-third]
  NEWSORT(A, ℓ + 2m, r)              [Sort last one-third]
  MERGE(A, ℓ, ℓ + 2m − 1, r)         [Merge first two-third with last one-third]
```

Write a recurrence relation for the worst-case running time of NEWSORT. Solve the recurrence relation to obtain a Θ-bound on the running time of the algorithm, in terms of the length n of the array. You may use any of the methods to solve the recurrence. You may also ignore floors and ceilings in your calculations.

Answer:
The recurrence for the worst-case running time of the above newsort variant is $T(m) \leq m + T(m/3) + T(2m/3)$.

Taking the base case T(1)=1
Now, solving the recurrence-
$T([]) \leq [] + T([]/3) + T(2[])/3$
Forming the recursion tree-



At all levels, the time is computed to be m. So, we get log(m)
For m levels , the net computation will be m(log(m))

The lower bound will be m(log$_3$(m)) due to T(m/3)

The upper bound will be m(log$_{3/2}$(m)) due to T(2m/3)

So the bound is
$m(log_3(m)) \leq m(log(m)) \leq m(log_{3/2}(m))$
So we get $T(n) = \Theta(m(log(m)))$

Problem 3. (4 points) Recurrences

Solve the following recurrences. You may assume for the base case that $T(1) = 1$.

(a) $T(n) = 8T(n/3) + n^2$. For convenience, you may assume that n is a power of 3.

Answer:
Solving the recurrence-
$T([]) \leq 8T([]/3) + []^2$

$T([]) \leq 8T(n/9) + n^2 + (n/3)^2$

$T([]) \leq 8T(n/27) + n^2 + (n/3)^2 + (n/9)^2$

So we get a series,

$T([]) \leq 8(n^2 + (n/3)^2 + (n/9)^2 + ...)$

$T([]) \leq 8 * n^2(1 + (1/3)^2 + (1/9)^2 + ...)$

$T([]) \leq 8 * n^2(1 + 1/9 + 1/81 + ...)$

Sum of this geometric progression is 1 / 1-r

r here is 1/2

So the sum is 1 / (1 - 1 /9)

$T(n) \leq 8 * (n)^2 * (9/8)$

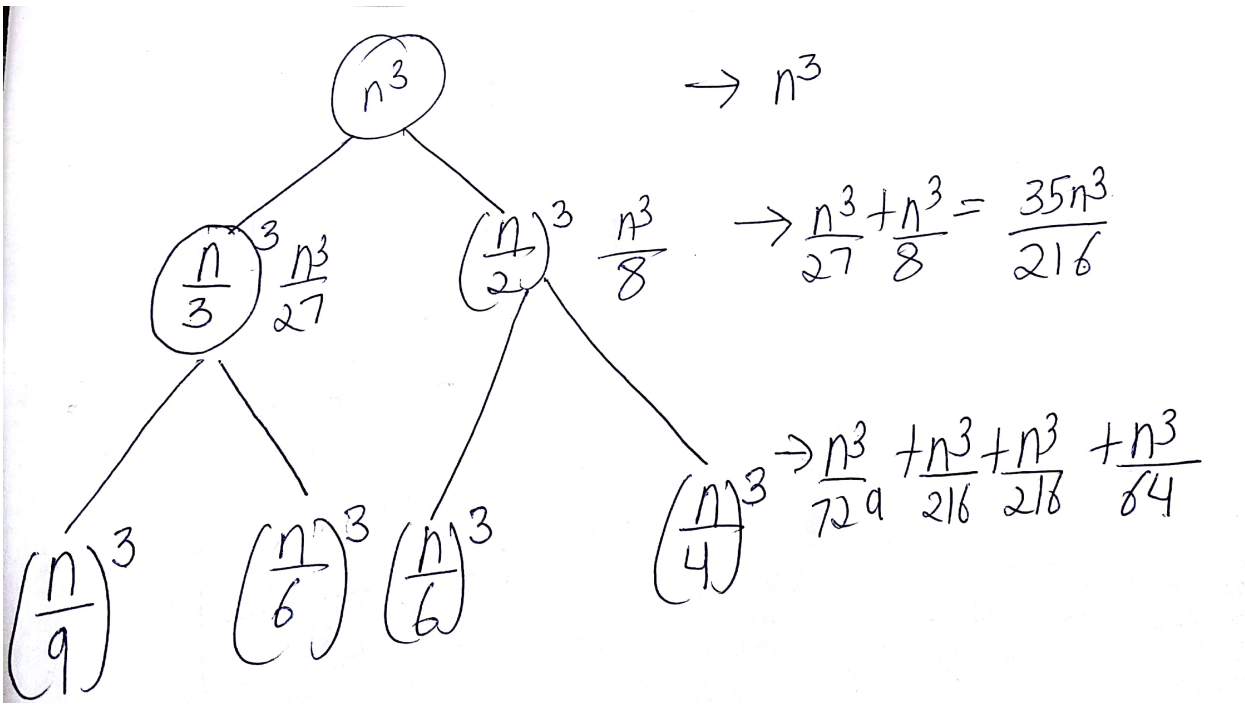
Therefore, $n^2 \leq T(n) \leq 9(n)^2$

$T(n) = \Theta(n^2)$

(b) $T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor n/2 \rfloor) + n^3$. For convenience, you may ignore floors and ceilings in your calculation.

Answer:
Solving the recurrence-
$T(n) \leq T([]/3) + T([]/2) + []^3$

Forming the recursion tree-



So we get a series,
$T(n) \leq n^3 + (n^3/27 + n^3/8) + (n^3/729 + n^3/216 + n^3/216 + n^3/64) + ...$

$T(n) \leq (n^3 + ((35 * n^3)/216) + (n^3/729 + n^3/108 + n^3/64) + ...$

$T(n) \leq (n^3 + ((35 * n^3)/216) + ((64 * 108)n^3 + (729 * 64)n^3 + (729 * 108)n^3)/(729 * 108 * 64)) + ...$

$T(n) \leq (n^3 + ((35 * n^3)/216) + (6912 + 46656 + 78732)n^3/(729 * 108 * 64) + ...)$

$T(n) \leq (n^3 + ((35 * n^3)/216) + (35 * 35 * 108)n^3/(216 * 216 * 108) + ...)$

$T(n) \leq (n^3 + ((35 * n^3)/216) + (35/216)^2 n^3 + ...)$

$T(n) \leq n^3(1 + 35/216 + (35/216)^2 + ...)$

Sum of this geometric progression is 1 / 1-r

r here is 35/216

So the sum is 1 / (1 - 35/216)

$T(n) \leq (n)^3 * (216/181)$

Therefore, $n^3 \leq T(n) \leq (n)^3 * (216/181)$

$T(n) = \Theta(n^3)$

Problem 4. (2 + 3 + 2 = 7 points) Finding good widgets using pairwise testing

You have n supposedly identical widgets that in principle can test each other. For instance, you can connect two widgets A and B, each of which then reports whether the other widget is good or bad, but the answer of a bad widget cannot be trusted. Thus, there are four possible outcomes of such a pairwise test.

Widget A says	Widget B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

You are told that more than $n/2$ of the widgets are good, and you are asked to determine all of the good widgets.

(a) Consider the problem of finding a single good widget. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.

Answer:
Assuming n to be even,
Let's take two widgets together for testing at a time.
If we get a result which is anything apart from the first case where either both are good or both are bad , we discard those widgets.
Out of those from the first case where they are either good or bad , we can take one widget.
Out of all the widgets that fall in the first case, the good widgets will be more as number of widgets that are good is more. -1
While removing in all the other cases , we are removing atleast one bad widget for one good widget. -2
Considering 1 and 2 we are left with a maximum of n/2 widgets and as the number of good widgets is more than those of the bad ones, at a minimum , half of them will be good in any scenario.
Hence, proved.
n/2 pairwise tests are sufficient to reduce the problem to one of nearly half the size if more than n/2 widgets are good.

(b) Using part (a), give a divide-and-conquer algorithm to find a single good widget using $\Theta(n)$ pairwise tests. Give and solve the recurrence that describes the number of tests.

Answer:

Divide(All widgets)
if(widgets==3)
  Test(widgets)
    if(both are good)
    pick any one widget to be true
    else
    The widget left is definitely a good one
    exit
else
  Take pairs of 2
  If case is (both are good)
    Pick one widget at random
    Add to new widget list
  If case is(!(both good))
    remove both widgets
Divide(new widget list)

Recursion is $T(n) \leq n/2 + T(n/2)$.
T(n/2) is for splitting the pair

n/2 is for testing

Now, solving the recurrence-
$T([]) \leq T([]/2) + []/2$

$T(n) \leq n/2 + T(n/2)$.

$T(n) \leq n/2 + n/4 + T(n/4)$.

$T(n) \leq n/2 + n/4 + n/8 + T(n/8)$.

So we get a series which is-

$T(n) \leq n/2 + n/4 + n/8 + n/16 + ...$

$T(n) \leq n/2(1 + 1/2 + 1/4 + 1/8 + ...)$

Sum of such geometric progression is 1 / 1-r

r here is 1/2
So the sum is 1 / (1 - 1 /2)
$T(n) \leq (n/2) * 2$

$T(n) \leq \Theta(n)$

So we get $T(n) = \Theta(n)$

(c) Using part (c), show that all the good widgets can be identified using $\Theta(n)$ pairwise tests.

Answer:
After we have found one good widget , we can use it to compare to find the other good widgets by doing pairwise testing one after the other.
This will lead to addition of n time to the time complexity

So we get $T(n) \leq n/2 + T(n/2) + n$

This is again equivalent to $\Theta(n)$.
Hence , proved.

2. (3 + 2 = 5 points) Optimal location of store

You are a new player in the mobile phone business and are planning to open a physical store in Manhattan, to rival Apple's well-established store there. You want to determine an optimal location to place the store. Manhattan is organized as a grid, and a simple model is to view it as an $m \times n$ grid consisting of points (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$. Each of the mn points constitutes both a potential location for the store as well as a neighborhood of homes. The distance between two

points (a, b) and (c, d) in Manhattan – $D((a, b), (c, d))$ – is simply $|a - c| + |b - d|$. For example, the distance between $(5, 3)$ and $(2, 9) = |5 - 2| + |3 - 9| = 9$.

After a detailed survey, you have found out which of the mn neighborhoods are really potential customers – so you have a set C of potential customers:

$$C = \{(i, j) : 1 \leq i \leq m, 1 \leq j \leq n, \text{ neighborhood at } (i, j) \text{ is a potential customer}\}.$$

For example, the set C could be $\{(1, 4), (1, 5), (1, 8), (2, 4), (3, 5), (6, 7), (2, 9)\}$. Since you have only one store to build, you would like to locate it at a point that minimizes the total distance from all the potential customers. That is, you want to find (x, y) in the grid that minimizes

$$\sum_{(i,j) \in C} D((i, j), (x, y)).$$

(a) Prove that an optimal location for the store is (x^*, y^*), where x^* is a median of $x_1, \ldots, x_n$ and y^* is a median of $y_1, \ldots, y_n$. That is, prove that for any (x, y):

$$\sum_{(i,j) \in C} D((i, j), (x^*, y^*)) \leq \sum_{(i,j) \in C} D((i, j), (x, y)).$$

Answer:
Proof-
Lets assume that the sum of distances from all the points to point x(not median) is minimum.)
Now, the sum of distances from all the points to x will increase as compared to the distance from median (x*) as the geometric median(x*) of a set of sample points in a space is defined to be the point minimizing the sum of distances to the other points.

Hence this contradicts our assumption that the sum of distances from the point x will be less than that from the median(x*) Hence by contradiction , a is true

(b) Using part (a), give an efficient algorithm that takes as input the $m \times n$ grid and the set C, and returns the optimal store location (x, y).

Answer: