Recitation 5 (HW4)

Online: Xinyi Zhao xz2833@nyu.edu

GCASL 475: Yifan Jin yj2063@nyu.edu

New York University

Basic Algorithms (CSCI-UA.0310-005)

Problem 1

Problem 1 (13+12 points)

Recall that in the deterministic approach for choosing the pivot in the SELECT algorithm, first we split the elements of the array into groups of size 5. We want to see how the running time of SELECT changes by varying the size of groups. Consider the following cases:

(a) Split the array into groups of size 3

(b) Split the array into groups of size 7

For each case, write the corresponding recursion for the running time of the SELECT algorithm, draw the corresponding recursion tree for the worst case, and use it to solve the recursion for the worst case.
For full credit, your answers must use $\Theta(.)$ notation (You do NOT need to use strong induction to prove your result).

Problem 1

How to get the recursion for running time?

Recall when array is splitted into groups of size 5.

$T(n) <= T(7n/10) + T(n/5) + O(n)$

Problem 1

How to get the recursion for running time?

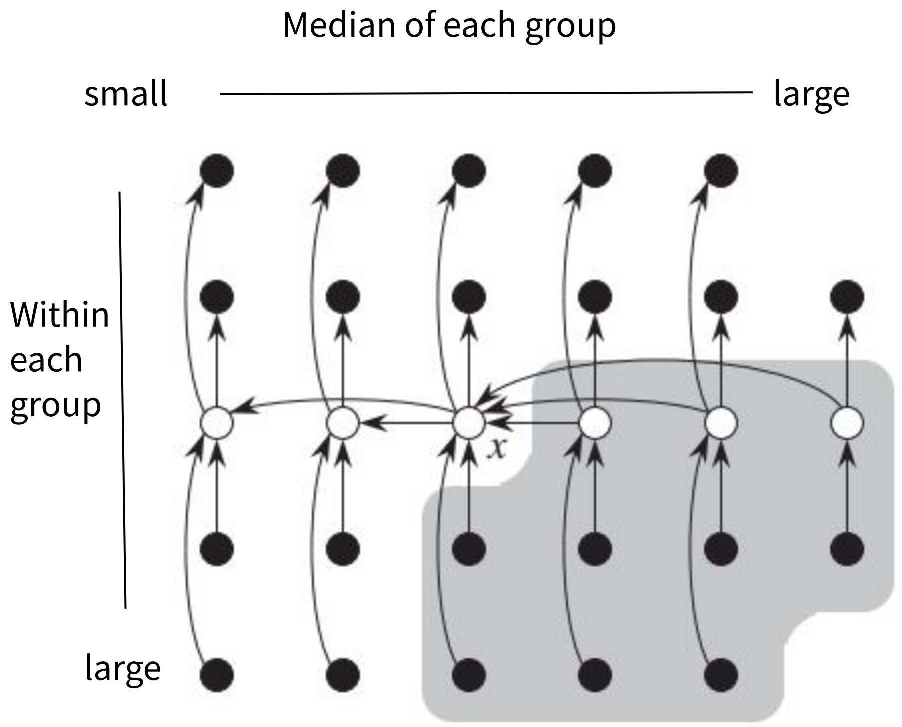
Recall when array is splitted into groups of size 5.

$T(n) <= \textbf{T(7n/10)} + T(n/5) + O(n)$

Find the median of the n/5 median of each group

Why is it?

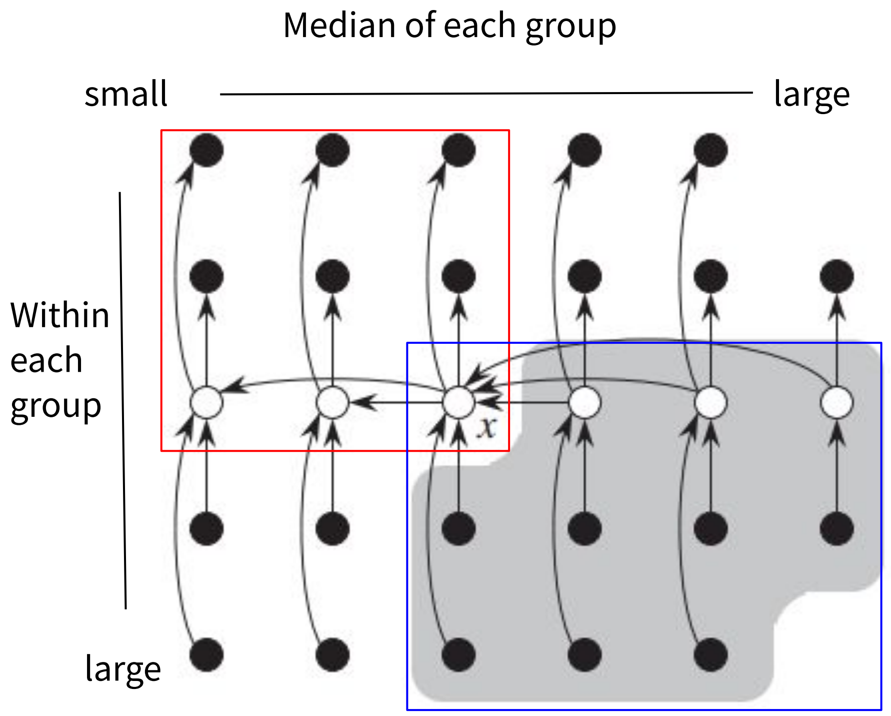
Problem 1



Assume x is the pivot,

Median of each group

Problem 1



Assume x is the pivot

ledian of median of each group

All elements in RED square must be

<= x, there are about **3/10 * n** elements

All elements in BLUE square must be

>= x, there are about **3/10 * n** elements

Problem 1

How to compute T(n)?

1. Find the median of each group of size x, there are n/x groups, each group takes O(1) time
2. Find the median of the newly formed array of medians, Select(array of size n/x, 2n/x)
3. Use this median as pivot, partition
4. Recursively calling over left or right part => what is the worst case??

Problem 1

How to compute T(n)?

Recursively calling over left or right part => what is the worst case??

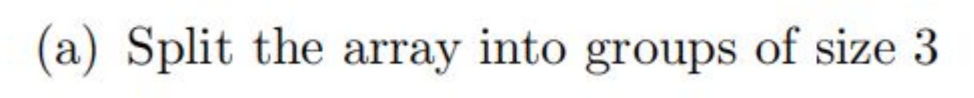
In each group (to the left of the pivot), there are >= (1+x/2) elements less than the pivot

There are n/(2x) groups (to the left of the pivot)

=>

At least (1+x/2) * n / (2x) elements less than the pivot

Problem 1


(a) Split the array into groups of size 3

$x = 3$

The recursion formula for running time:

$T(n) <= T(2/3 * n) + T(n/3) + n + n$

=>

$T(n) <= T(2/3 * n) + T(n/3) + n$

$T(1) = 1$

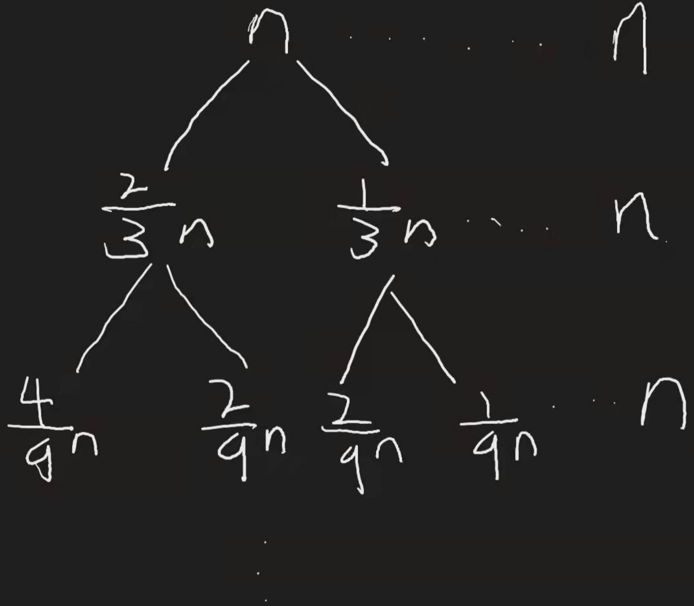
Problem 1

(a) Split the array into groups of size 3

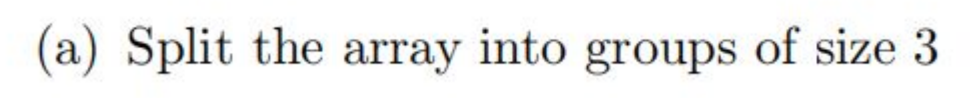
X = 3

T(n) <= T(2/3 * n) + T(n/3) + n

T(1) = 1



Problem 1


(a) Split the array into groups of size 3

How to solve the recursion?

Problem 1

(a) Split the array into groups of size 3

How to solve the recursion?

The rightmost branch has the shortest height, log n base 3.

So the lower bound of running time is a full binary tree with that height.

The running time of each layer is O(n).

Thus, T(n) = \Omega(n logn)

Problem 1

(a) Split the array into groups of size 3

How to solve the recursion?

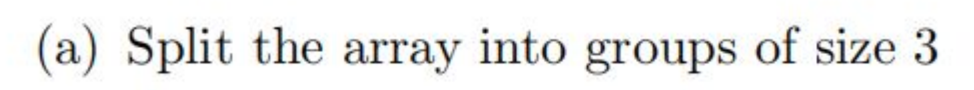
Similarly, the leftmost branch has the longest height, log n base 3/2.

So the upper bound of running time is a full binary tree with that height.

The running time of each layer is O(n).

Thus, T(n) = O(n logn)

Problem 1


(a) Split the array into groups of size 3

How to solve the recursion?

T(n) = \Omega(n logn)

T(n) = O(n logn)

Thus, T(n) = \theta(n logn)

Problem 1

(b) Split the array into groups of size 7

$X=7$

$T(n) \le T(5/7 * n) + T(n/7) + n$

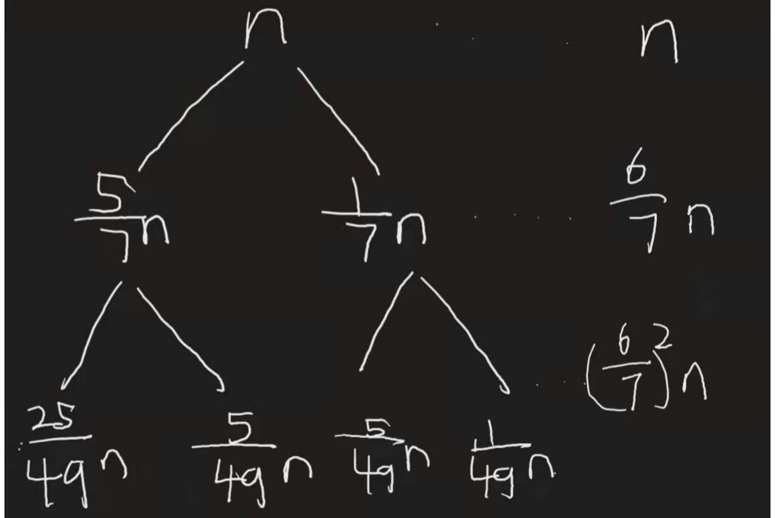
$T(1) = 1$

Problem 1

(b) Split the array into groups of size 7

$T(n) \le T(5/7 * n) + T(n/7) + n$

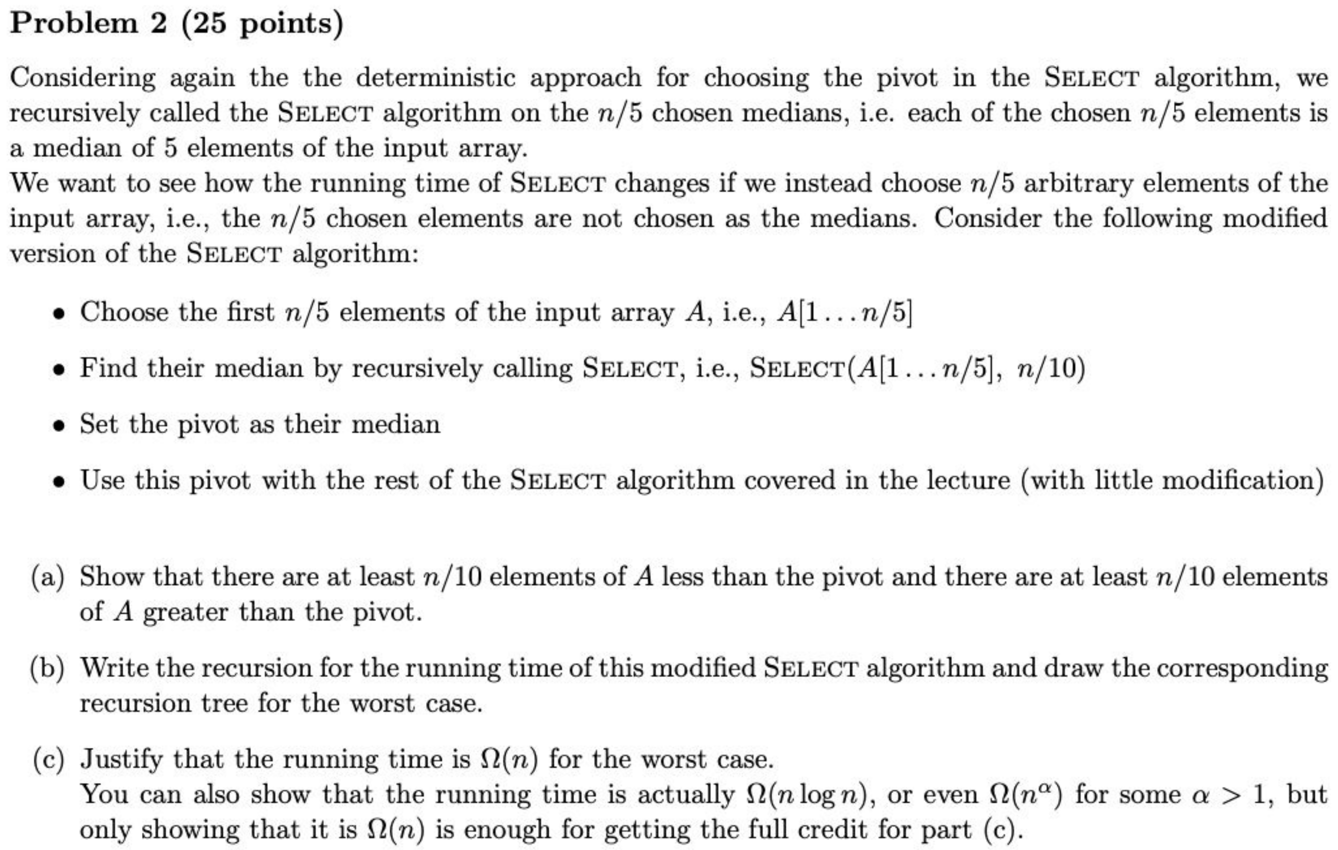
$T(1) = 1$



$$\text{sum} = \left[1 + \frac{6}{7} + \left(\frac{6}{7}\right)^2 + \dots\right] n$$

$$= \Theta(n)$$

Problem 2



Problem 2 (25 points)

Considering again the the deterministic approach for choosing the pivot in the SELECT algorithm, we recursively called the SELECT algorithm on the $n/5$ chosen medians, i.e. each of the chosen $n/5$ elements is a median of 5 elements of the input array.
We want to see how the running time of SELECT changes if we instead choose $n/5$ arbitrary elements of the input array, i.e., the $n/5$ chosen elements are not chosen as the medians. Consider the following modified version of the SELECT algorithm:

- Choose the first $n/5$ elements of the input array A, i.e., $A[1 \ldots n/5]$
- Find their median by recursively calling SELECT, i.e., SELECT$(A[1 \ldots n/5],\ n/10)$
- Set the pivot as their median
- Use this pivot with the rest of the SELECT algorithm covered in the lecture (with little modification)

(a) Show that there are at least $n/10$ elements of A less than the pivot and there are at least $n/10$ elements of A greater than the pivot.

(b) Write the recursion for the running time of this modified SELECT algorithm and draw the corresponding recursion tree for the worst case.

(c) Justify that the running time is $\Omega(n)$ for the worst case.
You can also show that the running time is actually $\Omega(n \log n)$, or even $\Omega(n^\alpha)$ for some $\alpha > 1$, but only showing that it is $\Omega(n)$ is enough for getting the full credit for part (c).

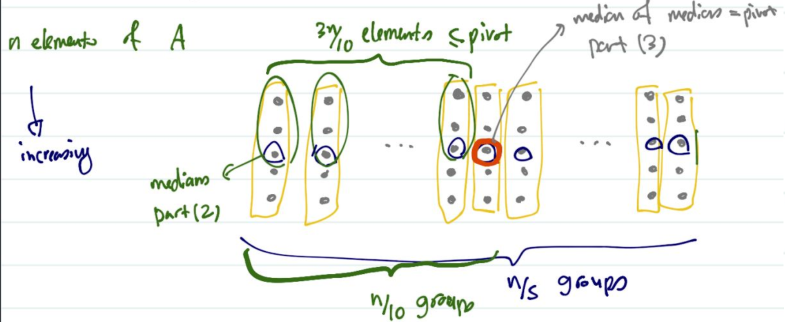
Problem 2

Recall Deterministic Approach

(1) Group n elements of A into n/5 groups each of size 5

(2) Find the median of each of the n/5 groups:
we get n/5 medians
use insertion sort to sort the 5 elem. in each group to find the median of each group

(3) Set the pivot as the median of the n/5 medians found in step (2): pivot := median of medians
recursively call Select to these n/5 elements



TC of Select using Method (2):

$$T(n) \lesssim \underbrace{T(7n/10)}_{\text{recursively calling over left or right part}} + \underbrace{n}_{\text{partition}} + \underbrace{\underbrace{T(n/5)}_{\text{Step (3) of Method (2)}} + \underbrace{n}_{\text{Step (2) of Method (2)}}}_{\text{TC of choosing the pivot using Step (2)}}$$

Problem 2

(a) Show that there are at least n/10 elements of A less than the pivot and there are at least n/10 elements of A greater than the pivot.

In A[1…n/5], how many elements are less than the pivot?

In A[1…n/5], how many elements are larger than the pivot?

Problem 2

(a) Show that there are at least n/10 elements of A less than the pivot and there are at least n/10 elements of A greater than the pivot.

In A[1…n/5], how many elements are less than the pivot? => n/10

So, in A[1…n], there are >= n/10 elements less than the pivot.

In A[1…n/5], how many elements are larger than the pivot? => n/10

So, in A[1…n], there are >= n/10 elements larger than the pivot.

Problem 2

(b) Write the recursion for the running time of this modified Select algorithm and draw the corresponding recursion tree for the worst case.

1. Choose the first n/5 elements of the input array A, i.e., A[1 . . . n/5] => $\theta(n/5) = \theta(n)$
2. Find their median by calling Select(A[1 . . . n/5], n/10) => $T(n/5)$
3. Set the pivot as their median, partition => $\theta(n)$
4. Use this pivot with the rest of the Select algorithm => $\leq T(9n/10)$

$T(n) \leq T(9n/10) + T(n/5) + n$

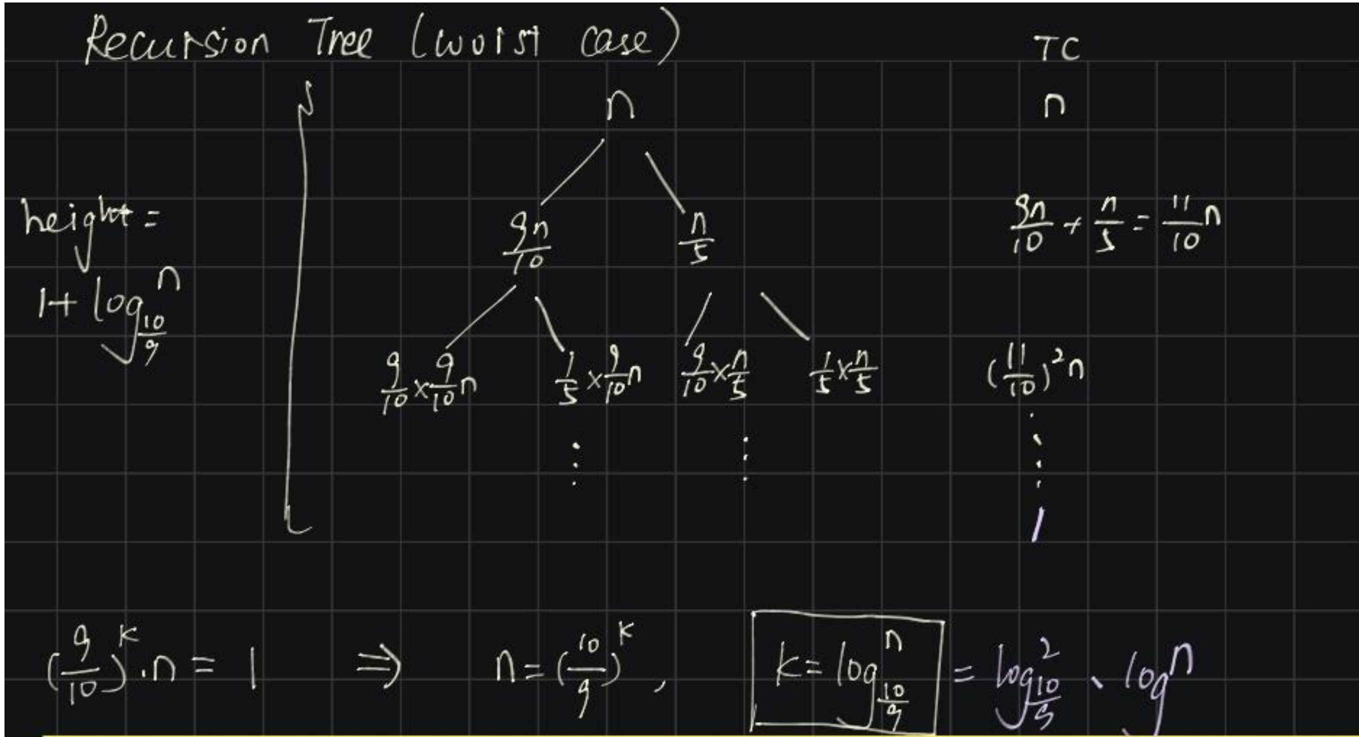
$T(1) = 1$

Problem 2

(b) Write the recursion for the running time of this modified Select algorithm and draw the corresponding recursion tree for the worst case.

$T(n) \leq T(9n/10) + T(n/5) + n$

$T(1) = 1$



Recursion Tree (worst case)
TC
n
n
$\frac{9n}{10}$
$\frac{n}{5}$
$\frac{9n}{10} + \frac{n}{5} = \frac{11}{10}n$
height = $1 + \log_{\frac{10}{9}} n$
$\frac{9}{10} \times \frac{9}{10} n$
$\frac{1}{5} \times \frac{9}{10} n$
$\frac{9}{10} \times \frac{n}{5}$
$\frac{1}{5} \times \frac{n}{5}$
$(\frac{11}{10})^2 n$
1
$(\frac{9}{10})^k \cdot n = 1 \Rightarrow n = (\frac{10}{9})^k, \boxed{k = \log_{\frac{10}{9}} n} = \log_{\frac{10}{9}}^{2} \cdot \log^{n}$

Problem 2

(c) Justify that the running time is Ω(n) for the worst case.

$T(n) \leq T(9n/10) + T(n/5) + n$

$T(1) = 1$

$$T(n) \cong n + \frac{11}{10}n + \left(\frac{11}{10}\right)^2 n + \cdots + 1$$

$$\geq n \underbrace{\boxed{\left(1 + \frac{11}{10} + \left(\frac{11}{10}\right)^2 \cdots\right)}}_{\downarrow\, c}$$

$$1 + \frac{11}{10} + \left(\frac{11}{10}\right)^2 + \cdots + \left(\frac{11}{10}\right)^i \quad , \quad i > 0$$

$$= \frac{1 - \left(\frac{11}{10}\right)^{i+1}}{1 - \frac{11}{10}}$$

$$= 10 \cdot \left(\left(\frac{11}{10}\right)^{i+1} - 1\right) > 0$$

$$T(n) \geq c \cdot n \quad , \quad c > 0$$

$$T(n) = \Omega(n)$$

Problem 3

Problem 3 (25 points)

We learnt how to solve recursions by using a recursion tree based approach. In this exercise, we learn another approach to solve recursions: using the *Master Theorem.*

Theorem 1. *(Master Theorem) Consider the following recursion*

$$T(n) = aT(n/b) + f(n),$$

with the constants $a \geq 1$ and $b > 1$ and the positive function $f(n)$ (i.e., $f(n) > 0$ for all n).
There are three cases:

(a) If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

(b) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

(c) If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large values of n, then $T(n) = \Theta(f(n))$.

Some remarks regarding the Master Theorem:

- The Master Theorem can be proved by drawing the corresponding recursion tree but we do not cover it for thc sakc of simplicity.
- Note that not all the recursions can be solved by using the Master Theorem (see Example III below).

Problem 3

For each of the following recursions, if it can be solved with the Master Theorem, use the Theorem to find the explicit answer for $T(n)$ (In this case, for full credit, your answers must use $\Theta(.)$ notation and you should check that all conditions of the Master Theorem apply as discussed in the Examples above). Otherwise, fully justify why the Master Theorem does not apply.

(a) $T(n) = 4T(n/2) + n^2$

(b) $T(n) = 2T(n/2) + \log n$

(c) $T(n) = 0.2T(n/2) + n$

(d) $T(n) = 8T(n/2) + 2^n$

(e) $T(n) = 2T(n/2) + \frac{n}{\log n}$

Problem 3

To simplify, the Master Theorem is just to compare two functions:

$$T(n) = aT(n/b) + f(n),$$

Compare $n^{\log_b a}$ with $f(n)$

Problem 3

To simplify, the Master Theorem is just to compare two functions:

$$T(n) = aT(n/b) + f(n),$$

Compare $n^{\log_b a}$ with $f(n)$

The asymptotically (polynomially) larger one is the final result.

If $f(n) = \Theta(n^{\log_b a})$, *then* $T(n) = \Theta(n^{\log_b a} \log n)$. Asymptotically equal

One more point: If f(n) is larger, then you also need to guarantee that

$$af(n/b) \leq cf(n) \text{ for some constant } c < 1$$

Problem 3

(a) $T(n) = 4T(n/2) + n^2$

Problem 3

(a) $T(n) = 4T(n/2) + n^2$

a = 4, b = 2, f(n) = n ^ 2

$n^{\log_b a}$ = n ^ 2

They are asymptotically equal, so the result is \theta(n^2 logn)

Problem 3

(b) $T(n) = 2T(n/2) + \log n$

Problem 3

(b) $T(n) = 2T(n/2) + \log n$

a = 2, b = 2,   f(n) = log n

$n^{\log_b a}$ = n is larger,   so the final result = \theta(n)

(In solution you need to give an \epsilon to prove)

Problem 3

(c) $T(n) = 0.2T(n/2) + n$

Problem 3

(c) $T(n) = 0.2T(n/2) + n$

a = 0.2, b = 2, f(n) = n.

Since Master Theorem requires **a>=1**,

hence Master Theorem does not apply.

Problem 3

(d) $T(n) = 8T(n/2) + 2^n$

Problem 3

(d) $T(n) = 8T(n/2) + 2^n$

a = 8, b = 2, f(n) = 2^n

$n^{\log_b a}$ = n^3, so f(n) is asymptotically larger.

Also, a*f(n/b) = 8 * 2^(n/2) <= c * 2^n when n is sufficiently large.

Thus, the final result = \theta(2^n)

Problem 3

(e) $T(n) = 2T(n/2) + \frac{n}{\log n}$

Problem 3

(e) $T(n) = 2T(n/2) + \frac{n}{\log n}$

a = 2, b = 2, f(n) = n / (log n)

$n^{\log_b a}$ = n, but the two are **not polynomially comparable.**

Problem 3

(e) $T(n) = 2T(n/2) + \frac{n}{\log n}$

A = 2, B = 2, f(n) = n / (log n)

$n^{\log_b a}$ = n, but the two are **not polynomially comparable.**

(a) If $f(n) = O(n^{\log_b a - \epsilon})$ *for some constant* $\epsilon > 0$*, then* $T(n) = \Theta(n^{\log_b a})$.

If **n / (log n) = O(n^(1-\epsilon))**, which means **n^\epsilon = O(log n)**

Such \epsilon does not exists, so the Master Theorem does not apply.

Problem 4

Problem 4 (12+13 points)

(a) Given the input array A of size n with distinct elements and the positive integer k satisfying $1 \leq k \leq n$, modify the SELECT algorithm to develop an $O(n)$ time algorithm to find all the kth smallest elements of A. In other words, your algorithm must output the 1st smallest, and the 2nd smallest, ..., and the kth smallest elements of A.
You should write the pseudo-code of your algorithm.

(b) Given the input array A of size n with distinct elements and the positive integers k, l, satisfying $1 \leq l \leq k \leq n$, develop an $O(n)$ time algorithm to find the lth smallest, and the $(l+1)$th smallest, ..., and the kth smallest elements of A (Thus, your algorithm must return $k - l + 1$ elements).
You do NOT need to write the pseudo-code of your algorithm.

Hint: Use part (a) to find all the kth smallest elements. Then, out of these k elements, we need to find all the $(k - l + 1)$th largest elements.

Problem 4

(a) Given the input array A of size n with distinct elements and the positive integer k satisfying $1 \leq k \leq n$, modify the Select algorithm to develop an O(n) time algorithm to find all the kth smallest elements of A. In other words, your algorithm must output the 1st smallest, and the 2nd smallest, . . . , and the k-th smallest elements of A.

After calling the original QuickSelect(A[1…n], k), what does the array look like?

=> A[1…k…n]

Problem 4

(a) Given the input array A of size n with distinct elements and the positive integer k satisfying 1 ≤ k ≤ n, modify the Select algorithm to develop an O(n) time algorithm to find all the kth smallest elements of A. In other words, your algorithm must output the 1st smallest, and the 2nd smallest, . . . , and the k-th smallest elements of A.

After calling the original QuickSelect(A[1…n], k), what does the array look like?

=> A[1…k…n]

A[1…k] <= A[k]

A[k+1…n] >= A[k]

Problem 4

(a) Given the input array A of size n with distinct elements and the positive integer k satisfying 1 ≤ k ≤ n, modify the Select algorithm to develop an O(n) time algorithm to find all the kth smallest elements of A. In other words, your algorithm must output the 1st smallest, and the 2nd smallest, . . . , and the k-th smallest elements of A.

After calling the original QuickSelect(A[1…n], k), what does the array look like?

=> A[1…k…n]

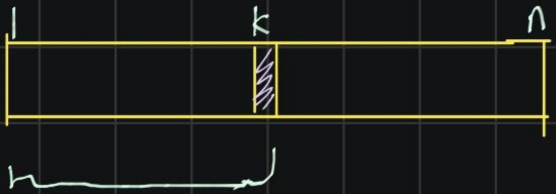
A[1…k] <= A[k]   => A[1…k] contains k elements which are the top k smallest elements.

A[k+1…n] >= A[k]

Problem 4

(a) Given the input a e integer k satisfying 1 ≤ k ≤ n, n lgorithm to find all the kth small ust output the 1st smallest, and the A.

```
Select (A[1...n], k)
    choose a pivot from A[1...n]
    pivot = A[i]
    j = partition (A[1...n], pivot = A[i])
    if k == j :
        $\boxed{\text{return } k}$        (return the index)
    else if k < j :
        return Select (A[1...j-1, k])
    else :
        return Select (A[j+1...n], k-j)
```



$A[1 \cdots k] \leq A[k]$

Problem 4

(b) Given the input array A of size n with distinct elements and the positive integers k,l, satisfying $1 \le l \le k \le n$, develop an O(n) time algorithm to find the l-th smallest, and the (l + 1)-th smallest, . . . , and the kth smallest elements of A (Thus, your algorithm must return $k - l + 1$ elements).

Hint: Use part (a) to find all the kth smallest elements.

Problem 4

(b) Given the input array A of size n with distinct elements and the positive integers k,l, satisfying $1 \le l \le k \le n$, develop an O(n) time algorithm to find the l-th smallest, and the (l + 1)-th smallest, . . . , and the kth smallest elements of A (Thus, your algorithm must return $k - l + 1$ elements).

After calling Select(A[1…n], k), we get:

A[1..k] which are the top k smallest element in A.

Problem 4

(b) Given the input array A of size n with distinct elements and the positive integers k,l, satisfying $1 \le l \le k \le n$, develop an O(n) time algorithm to find the l-th smallest, and the (l + 1)-th smallest, . . . , and the kth smallest elements of A (Thus, your algorithm must return $k - l + 1$ elements).

How to divide A[1…k] into two parts: A[1…l-1], A[l…k]

S.t. A[l…k] will contain the l-th smallest, and the (l + 1)-th smallest, . . . , and the kth smallest elements

=> We can call Select(A[1…k], l)

Problem 4

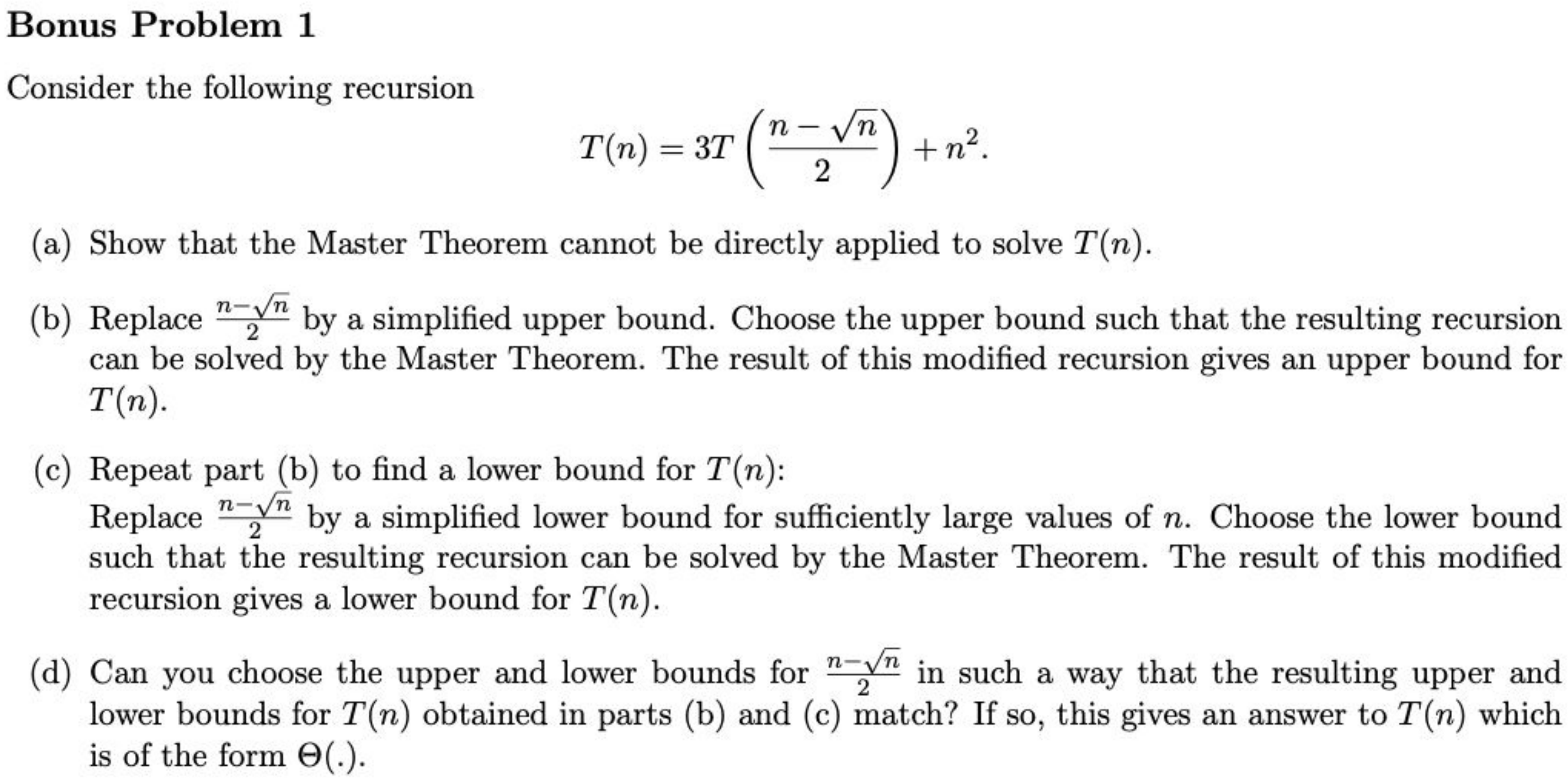
(b) Given the input array A of size n with distinct elements and the positive integers k,l, satisfying $1 \le l \le k \le n$, develop an O(n) time algorithm to find the l-th smallest, and the (l + 1)-th smallest, . . . , and the kth smallest elements of A (Thus, your algorithm must return $k - l + 1$ elements).

Call Select(A[1…k], l),

then we get A[1…l…k],

we can say, A[l…k] is the desired result

Bonus Problem 1



Bonus Problem 1

Consider the following recursion

$$T(n) = 3T\left(\frac{n-\sqrt{n}}{2}\right) + n^2.$$

(a) Show that the Master Theorem cannot be directly applied to solve $T(n)$.

(b) Replace $\frac{n-\sqrt{n}}{2}$ by a simplified upper bound. Choose the upper bound such that the resulting recursion can be solved by the Master Theorem. The result of this modified recursion gives an upper bound for $T(n)$.

(c) Repeat part (b) to find a lower bound for $T(n)$:
Replace $\frac{n-\sqrt{n}}{2}$ by a simplified lower bound for sufficiently large values of n. Choose the lower bound such that the resulting recursion can be solved by the Master Theorem. The result of this modified recursion gives a lower bound for $T(n)$.

(d) Can you choose the upper and lower bounds for $\frac{n-\sqrt{n}}{2}$ in such a way that the resulting upper and lower bounds for $T(n)$ obtained in parts (b) and (c) match? If so, this gives an answer to $T(n)$ which is of the form $\Theta(.)$.

Bonus Problem 1

(a)

$$T(n) = 3T\left(\frac{n-\sqrt{n}}{2}\right) + n^2$$

$$T\left(\frac{n-\sqrt{n}}{2}\right) \not\Leftrightarrow T(n/b)$$

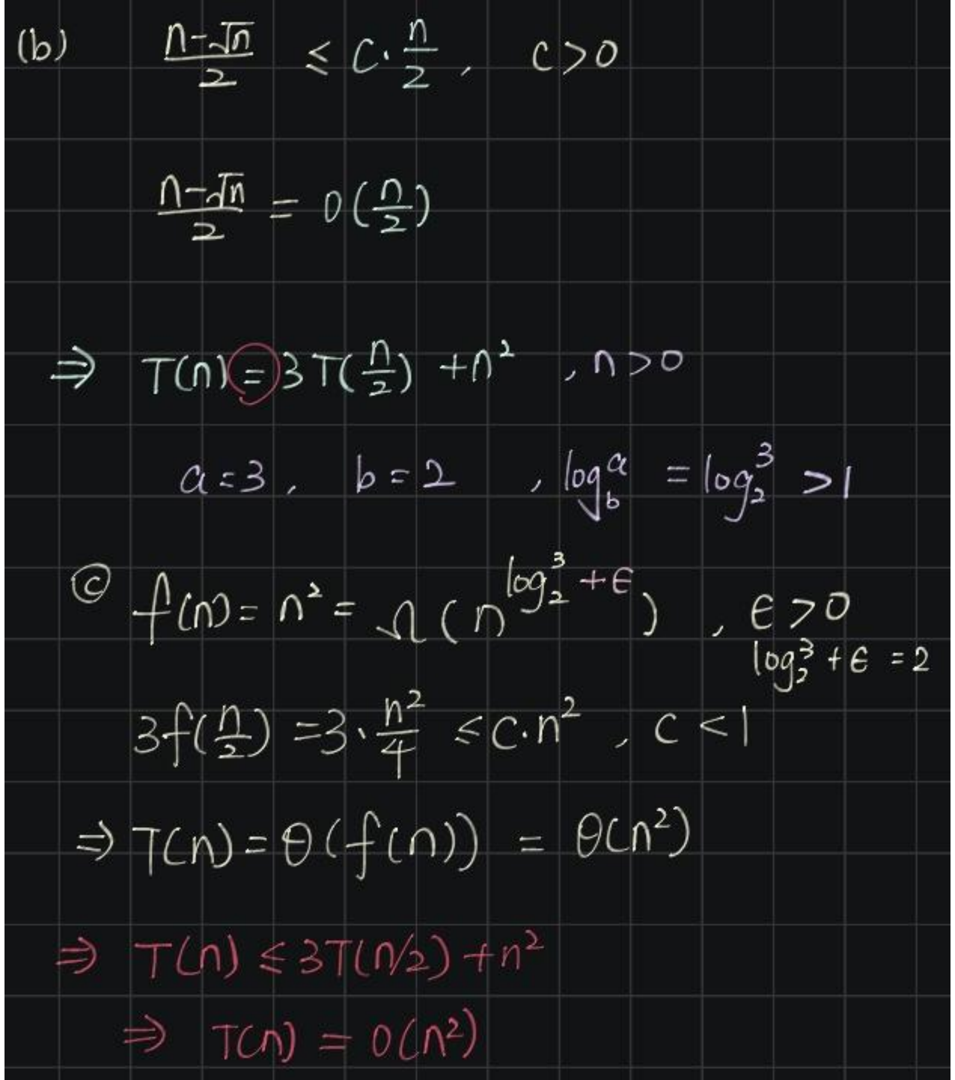
$$T(n) = aT(n/b) + f(n)$$

$a \geq 1$

$b > 1$

$f(n) > 0$

Bonus Problem 1



(b) $\frac{n-\sqrt{n}}{2} \le c\cdot\frac{n}{2}$, $c>0$

$\frac{n-\sqrt{n}}{2} = O(\frac{n}{2})$

$\Rightarrow T(n) = 3T(\frac{n}{2}) + n^2$, $n>0$

$a=3$, $b=2$, $\log_b^a = \log_2^3 > 1$

ⓒ $f(n) = n^2 = \Omega(n^{\log_2^3+\epsilon})$, $\epsilon > 0$

$\log_2^3 + \epsilon = 2$

$3f(\frac{n}{2}) = 3\cdot\frac{n^2}{4} \le c\cdot n^2$, $c<1$

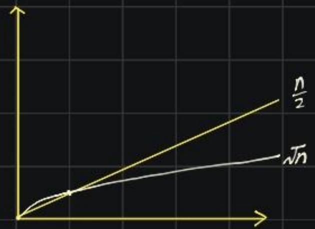
$\Rightarrow T(n) = \Theta(f(n)) = \Theta(n^2)$

$\Rightarrow T(n) \le 3T(n/2) + n^2$

$\Rightarrow T(n) = O(n^2)$

Bonus Problem 1

(c) $\frac{n-\sqrt{n}}{2}$



$n>4,\ \sqrt{n} < \frac{n}{2}$

$\Rightarrow \frac{n-\sqrt{n}}{2} > \frac{n-\frac{n}{2}}{2} = \frac{n}{4}$

$\frac{n-\sqrt{n}}{2} = \Omega(\frac{n}{4})$

$\Rightarrow T(n) \enclose{circle}{=} 3T(\frac{n}{4}) + n^2,\ n>0$

$a=3,\ b=4,\ \log_b^a = \log_4^3 < 1$

$f(n) = n^2 = \Omega(n^{\log_4^3+\epsilon}),\ \epsilon>0,\ \log_4^3+\epsilon=2$

$3f(n/4) = 3\cdot\frac{n^2}{16} \le c\cdot n^2,\ c<1$

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

$\Rightarrow T(n) \geq 3T(\frac{n}{4}) + n^2$

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

Bonus Problem 1

(d)

From (b), $T(n) = O(n^2)$

From (c), $T(n) = \Omega(n^2)$

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

Q & A

Thank you