COMP 3711 – Design and Analysis of Algorithms
2021 Fall Semester – Written Assignment # 2
Distributed: Sept 23, 2021
Due: Oct. 4, 2021, 11:59
Corrected Sept 27, 2021
Solution

Your solution should contain

(i) your name, (ii) your student ID #, and (iii) your email address

at the top of its first page.

Some Notes:

- Please write clearly and briefly. Your solutions should follow the guidelines given at
  https://canvas.ust.hk/courses/38226/pages/assignment-submission-guidelines

  In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.

- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page.
  You must acknowledge individuals who assisted you, or sources where you found solutions. Failure to do so will be considered plagiarism.

- The term *Documented Pseudocode* means that your pseudocode must contain documentation, i.e., comments, inside the pseudocode, briefly explaining what each part does.

- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.

- Please make a *copy* of your assignment before submitting it. If we can't find your submission, we will ask you to resubmit the copy.

- Submit a SOFTCOPY of your assignment to Canvas by the deadline. The softcopy should be one PDF file (no word or jpegs permitted, nor multiple files).

  If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

- Sept, 27, 2021. Typo in Problem 2 (3) definition on p 5 corrected.
  $\{(x, y) : 1 \leq x \leq n, 1 \leq y \leq n\}$ was fixed to be $\{(x, y) : 1 \leq x \leq m, 1 \leq y \leq m\}$.

Problem 1: [30 pts]

Let A be an array of n elements. A *heavy element* of A is any element that appears more than 3/20 times, i.e., is more than 15% of the items. For example, if $n = 14$ then a heavy element is one that appears at least $3 = \left\lceil \frac{3}{20} \cdot 14 \right\rceil$ times.

As examples, consider the arrays A, B and C below:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$A[i]$	2	6	3	5	6	3	8	8	2	7	4	9	9	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$B[i]$	3	3	3	5	6	3	8	3	6	7	4	9	9	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$C[i]$	3	3	3	6	6	3	8	3	6	6	6	9	9	9

A contains no heavy items. B contains the unique heavy item 3. C contains the three heavy items 3, 6 and 9.

Design an $O(n \log n)$ time **divide-and-conquer algorithm** for finding all the heavy items (if any) in an array.

Your solution should be split into the following three parts. Write the answer to each part separately.

(a) (i) Write documented pseudocode for a procedure $Heavy(i, j)$ that returns the *set* of heavy items in subarray $A[i \ldots j]$.

(ii) Below the pseudocode, provide a description in words (as opposed to code) of what your algorithm is doing.

(b) Explain (prove) why your algorithm is correct.

(c) Let $T(n)$ be the worst case number of total operations of all types performed by your algorithm. Derive a recurrence relation for $T(n)$ (explain how you derived the recurrence relation).

Show that $T(n) = O(n \log n)$.

Note: If the recurrence relation is in the form

$$\forall n > 1, \quad T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_1 n \quad \textit{and} \quad T(1) = c_2 \qquad (1)$$

for some $c_1, c_2 \geq 0$ you may immediately conclude that $T(n) = O(n \log n)$ without providing any further explanation other than saying you are calling the Master Theorem (and explaining which case of the master theorem it is and why).

Otherwise, you must prove from first principles that whatever recurrence relation you derived implies $T(n) = O(n \log n)$ for all values of n.

See below for requirements, hints and general comments.

Requirements, Hints and Comments:

- *Hint: The solution to Tutorial problem DC12 will help you get started.*
- *Hint: Let $S = Heavy(i, j)$. Prove that subarray $A[i \ldots j]$ cannot contain more than 6 heavy items. You will need this fact in your analysis*
- **Sorting is not permitted.**
  The only comparisons allowed **to items in the array** are equalities. Inequalities are not permitted. More specifically:
  - You MAY write "If $A[i] = A[j]$" or "If $A[i] = x$".
  - You MAY NOT write "If $A[i] < A[j]$" or "If $A[i] < x$".
    Note that this immediately implies that you can NOT sort the array (since sorting requires inequality comparisons).
  - In particular you MAY NOT solve the problem by sorting the array and then counting all of the items with the same value.
- Part (a) requires two different types of explanations as to what your algorithm is doing. The first is the documentation IN the pseudocode. The second is the explanation AFTER the pseudo-code. BOTH must be provided.
- Part (b) is a proof of correctness. Write it as you would a mathematical proof. Explicitly state any facts upon which the proof depends. Incomplete proofs will have points deducted. See the Practice Homework on the Tutorial page for pointers as to how this should be structured.
- Answer Parts (a) and (b) separately. Do not worry if your explanation in Part (a) and your proof of correctness in Part (b) overlap somewhat.
- We are specifically asking for a divide-and-conquer algorithm here, e.g., a generalization of DC12. We are aware that faster non divide-and-conquer algorithms do exist and you might be able to find one by Googling. Such algorithms will not be accepted as answers. The purpose of this exercise is to use the divide-and-conquer approach to solve the problem.
- The call $Heavy(1, n)$ should run in $O(n \log n)$ time and return a *set*. You can do this by writing $Return(S)$, where S is a set.
- *Set Notation:* Your pseudocode will need to use a *set data structure* with associated *set operations*. You should use standard mathematical notation for writing set operations (do not make up your own set notation or use a notation from a programming language you know).

  See the next page for more information on how to write set operations in pseudocode along with associated running times.

Set notation in pseudocode: In what follows, S, S_1, S_2 are all sets.

Recall that sets do not contain repeated items.
So, if $S_1 = \{a, b\}$, $S_2 = \{b, c\}$ and $S = S_1 \cup S_2$, then $S = \{a, b, c\}$.

$|S|$ denotes the size of S, e.g, $|S_1| = |S_2| = 2$, $|S_1 \cup S_2| = 3$, $|S_1 \cap S_2| = 1$.
If $S = \emptyset$ (the empty set), then $|S| = 0$.

"$\cup$" denotes the union of sets. '$\cap$" denotes the intersection of sets.

$S_1 \setminus S_2$ (equivalently, $S_1 - S_2$) is the set S_1 with all items in $S_1 \cap S_2$ removed.

- Create sets using the notation "`Create Set` S".
  This creates empty set S in $O(1)$ time.
- Evaluating $|S|$ requires $O(1)$ time.
- Setting $S_1 = S_2$ will require $O(|S_2| + 1)$ time.
- You can add element x to set S by writing $S = S \cup \{x\}$.
- Evaluating $S = S_1 \cup S_2$, $S = S_1 \cap S_2$ and $S = S_1 \setminus S_2$ all require $O((|S_1| + 1)(|S_2| + 1))$ time.
- Evaluating the statement "$x \in S$" requires $O(|S|+1)$ time and returns TRUE if x is in S and FALSE otherwise.
- You may write pseudocode in the form

  $\forall x \in S$ do
  $Work(x)$

  where $Work(x)$ is some code dependent upon x.

  If $S = \{x_1 \ldots, x_k\}$, the time required to run this code will be $O(1)$ plus the total of the work required to run all of $Work(x_1)$, $Work(x_2), \ldots, Work(x_k)$.

Example 1: The following code finds the unique items in array A:

```
Create Set S
For i = 1 to n do
    S = S ∪ {A[i]}
```

It uses $O(n \cdot m)$ time, where m is the number of unique items in $A[1 \ldots n]$.

Example 2: The following code constructs $S_1 \cap S_2$:

```
Create Set S
∀x ∈ S1 do
    If x ∈ S2 then
        S = S ∪ {x}
```

It runs in $O((|S_1| + 1)(|S_2| + 1))$ time.

Solution.

(a)

Algorithm 1 (i) Heavy(i, j) : Returns set of heavy items in subarray $A[i \dots j]$

```
1:  Create Set S, S_1, S_2, R
2:  if j = i then                                   ▷ Termination Condition
3:      Return({A[i]})                              ▷ Unique item is heavy item
4:  else
5:          ▷ Recursively find the heavy items in left and right half subarrays
6:                ▷ Check each separately to check if it is heavy in A[i...j].
7:
8:      q = ⌊(i+j)/2⌋                               ▷ q is the midpoint of A[i...j]
9:      S_1 = Heavy(i, q)                           ▷ |S_1| ≤ 6
10:     S_2 = Heavy(q + 1, j)                       ▷ |S_2| ≤ 6
11:     S = S_1 ∪ S_2;                              ▷ S contains potential heavy items
12:
13:     for all x ∈ S do                            ▷ Check if x is heavy in A[i...j].
14:         count = 0
15:         for k = i to j do
16:             if A[k] = x then
17:                 count = count + 1
18:         if count > (3/20)(j − i + 1) then
19:             R = R ∪ {x}                         ▷ R stores heavy elements in A[i...j]
20:     Return(R)
```

(ii)

- If subarray has size 1 ($i = j$) then it reports the element in that subarray;
- If subarray has size > 1 algorithm splits it into left and right halves
  - It recursively finds the sets of heavy items in the left half (S_1) and right half (S_2).
  - It then checks each of the items in $S = S_1 \cup S_2$ and reports the subset $\bar{S}$ of them that are heavy in the full subarray.
- Note that if $S_1 \cap S_2 \neq \emptyset$ the operation $S = S_1 \cup S_2$ on line 10 removes duplicate items.
- Note if $S_1 \cup S_2 = \emptyset$, then line 13 is not implemented because no $x \in S$ exists.

(b) Correctness will use the following lemma:

Lemma: If x is heavy in subarray $A[i \ldots j]$ and $q \in [i \ldots j]$ then x must be heavy in at least one of $A[i \ldots q]$ and $A[q+1 \ldots j]$.

Proof: We will prove the contrapositive, i.e., that if x is not heavy in either of $A[i \ldots q]$ and $A[q+1 \ldots j]$, then it is not heavy in $A[i \ldots j]$.

Suppose x is not heavy in either of $A[i \ldots q]$ and $A[q+1 \ldots j]$. Then

$$\begin{aligned} |\{k \,:\, i \le k \le q, \text{ and } A[k] = x\}| &\le \frac{3}{20}(q-i+1) \\ |\{k \,:\, q+1 \le k \le j, \text{ and } A[k] = x\}| &\le \frac{3}{20}(j-q). \end{aligned}$$

Combining these yields

$$|\{k \,:\, i \le k \le j, \text{ and } A[k] = x\}| \le \frac{3}{20}(q-i+1) + \frac{3}{20}(j-q) \le \frac{3}{20}(j-i+1),$$

so x is not heavy in $A[i \ldots j]$. ■

The lemma immediately implies that if

$$\begin{aligned} S_1 &= \text{the set of heavy elements of } A[i \ldots q] \\ S_2 &= \text{the set of heavy elements of } A[q+1 \ldots j] \\ R &= \text{the set of heavy elements of } A[i \ldots j] \end{aligned}$$

then $R \subseteq S_1 \cup S_2$.

Note that it is quite possible that any or all of $S_1, S_2, \bar{S}$ are empty.

The correctness of the algorithm now follows by induction.

Let $m = j - i + 1$ be the *size* of subarray $A[i \ldots j]$.

- The algorithm is definitely correct when $m = 1$.
- The induction hypothesis is that the algorithm is correct for all problems of size $m' < m$.
- To see correctness for m, note that, from the induction hypothesis, the algorithm correctly finds S_1 and S_2. It then (lines 14-18) individually checks each item in $S_1 \cup S_2$ to see if they are heavy. The lemma implies that it therefore correctly finds all heavy elements in the full subarray.

(c) When calling Heavy(i, j), let $m = j - i + 1$ be the *size* of subarray $A[i \dots i]$.

If an element is heavy, then it appears more than $\frac{3}{20}m$ times in $A[i \dots j]$. Thus, the number of heavy items in any array is at most 6, since otherwise the array of size m would contain more than $7 \cdot \frac{3}{20}m > m$ copies of its heavy elements!

If $m = 1$ then $O(1)$ work is performed (Line 2 of the code). So $T(1) = O(1)$.
If $m > 1$ then the algorithm performs

- $T(m_1)$ work on line 9 where $m_1 = q - i + 1 = \lceil m/2 \rceil$.
- $T(m_2)$ work on line 10 where $m_2 = j - q = \lfloor m/2 \rfloor$
- $O(1)$ work on lines 11 and 12.
  Since $|S_1| \leq 6$, and $|S_2| \leq 6$ the time to perform $S = S_1 \cup S_2$ is $O((|S_1| + 1)(|S_2| + 1)) = O(49) = O(1)$.
- For any fixed x, Lines 15-20 require $O(m)$ time.
  If $|S| = 0$, lines 14-20 require $O(1)$ time (just to check that S is empty).
  If $S| > 0$ then lines 14-20 require $O(|S|m)$ time. Since $|S| \leq 12$, this requires $O(m)$ time.

Thus $T(1) = O(1)$ and

$$T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n).$$

Marking Note. The recursion could have been stopped earlier.
If the subarray has size $m = j - i + 1 \leq 6$ then $1 \geq \frac{3}{20}m$ so every item in the subarray is heavy.
This implies that you could stop the recursion when $m \leq 6$ and just report that every item is heavy.

Marking Note 3(a) If $S_1 \cap S_2 \neq \emptyset$ intersect then the operation (line 11) $S = S_1 \cup S_2$ checks for and removes the repeated items so that S only contains one copy of each item

That check for repeated items is what the $O((|S_1|+1)(|S_2|+1))$ is doing and is what keeps $|S| \leq 12$.

If you did NOT use the set operations as described but instead hard coded your own set operations you needed to explicitly describe the fact that you removed repeats. If you did use the set operations, then, for full credit, you needed to explicitly reference the cost of the set operations and the fact that $|S| \leq 12$ (or at least uniformly bounded).

- **If you did not EXPLICITLY remove repeats, then your algorithms would either not work or not work in $O(n \log n)$ time.**
- **Consider an array of size $n = 2^k$ that is composed of repeated $1, 2$s, i.e., $A = [1, 2, 1, 2, 1, 2 \ldots]$.**

  Then every recursive call (except the terminating ones) should always return heavy elements $1, 2$.
- **If your algorithm returned a list then it would have have to check the two lists returned by the recursive calls, notice that 1 and 2 were contained in both and remove one of each.**
- **If you did not do that explicit removal then your list could grow large, at the end containing $n/2$ 1s and $n/2$ 2s. The algorithm would then not be $On \log n)$,**
- **If you just assumed that your returned list was at most size 2 then your algorithm would run in $O(n \log n)$ time but it would be wrong, because you didn't explicitly say which items were removed and how.**

Marking Note 3(b)

Any proof of correctness had to prove the given Lemma or something equivalent. At the very least, explain why the fact in the Lemma is true. You could not just state the fact without proof.

Marking Note 3(c)

It was necessary to explicitly note that at each step $|S_1|, |S_2| \leq 6$, which is what permits the $O(n)$ combine step.

Problem 2 [28 pts] **Indicator Random Variables**

$A[1 \ldots n]$ is an array. In addition to A you are given 2 other structures that are built from A: a binary tree T and a $m \times m$ 2D matrix M. In each of these structure we also define $N(v)$, the set of neighbors of item v. Each of these structures and their neighbor definitions are defined below. See the diagram on the next page for examples.

(1) Array A: This is just the standard array.
Neighbors are the items to the left and right of i.

$$N(i) = \begin{cases} \{i-1, i+1\} & \text{if } 1 < i < n \\ \{2\} & \text{if } i = 1 \\ \{n-1\} & \text{if } i = n \end{cases}$$

(2) Binary Tree T :
In this case we assume that $n = 2^k - 1$ for some integer $k \geq 0$.
The correspondence between A and T is given below (and is also in the Heapsort slides page 9 onwards. Please see those for more explanation).
Each node in the tree corresponds to some element of array A. Node i will have value $T[i] = A[i]$.

The root of the tree is node 1, and has no parent.
The parent of node $i \neq 1$ is node $\lfloor i/2 \rfloor$. The left and right children of node i are nodes $2i$ and $2i + 1$, if they exist

$$N_T(i) = \begin{cases} \{2, 3\} & \text{if } v = 1 \\ \{2i, 2i+1, \lfloor i/2 \rfloor\} & \text{if } 1 < i < (n+1)/2 \\ \{\lfloor i/2 \rfloor\} & \text{if } (n+1)/2 \leq i \leq n \end{cases}$$

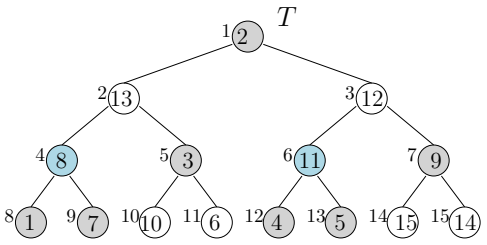
(3) $m \times m$ 2D matrix M.
In this case we assume that $n = m^2$.
$M(i, j) = A[(i-1)m + j]$. An element's neighbors are its (at most 4) vertical and horizontal neighbors. That is,

$$N_M((i,j)) = \{(i+1, j),\ (i-1, j),\ (i, j+1),\ (i, j-1)\} \bigcap \{(x, y) \ :\ 1 \leq x \leq m,\ 1 \leq y \leq m\}\,.$$

Now solve the six subproblems below. See the end of this problem for an explanation of what "Derive the expected number" means and how your solutions should be structured.

Local minima are shaded gray; saddle points, light blue.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$A[i]$	2	13	12	8	3	11	9	1	7	10	6	4	5	15	14



A has 5 local minima and 6 saddle points;
T has 7 local minima and 2 saddle points.

---

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$A[i]$	16	6	14	7	11	10	13	4	1	8	15	9	3	2	12	5

$i \backslash j$	1	2	3	4
1	16	6	14	7
2	11	10	13	4
3	1	8	15	9
4	3	2	12	5

M

A has 6 local minima and 4 saddle points; M has 5 local minima and 3 saddle points.

For all of the six problems assume that A is a uniform random permutation of $< 1, 2, \ldots, n >$. This means that every one of the $n!$ possible permutations is equally likely to occur.

The corresponding T and M are built from A as previously described.

The solutions to all six of the problems must use the indicator random variable technique to derive the result.

(a) **Counting local minima.**

(i) $A[i]$ is a *local minimum* in A if it is smaller than all of its neighbors, i.e, $A[i] < \min_{v \in N(i)} A[v]$.
Derive the expected number of local minima in A.
Hint: Let X_i be the indicator random variable for the event that $A[i]$ is a local minimum. What is $E(X_i)$ (this might depend upon i)? How does knowing the $E(X_i)$ let you answer the problem?

(ii) $T[i]$ is a *local minimum* in T if it is smaller than all of its neighbors, i.e, $T[i] < \min_{v \in N_T(i)} T[v]$.
Derive the expected number of local minima in T.

(iii) $M(i, j)$ is a *local minimum* in M if it is smaller than all of its neighbors, i.e, $M(i, j) < \min_{(x,y) \in N_M(i,j)} M(x, y)$.
Derive the expected number of local minima in M.

(b) **Counting Saddle points.**

(i) $A[i]$ is a *saddle point* in A if $i \neq 1$, $i \neq n$, and $A[i]$ is larger than one of its neighbors and smaller than the other one of its neighbors.
Derive the expected number of saddle points in A.

(ii) $T[i]$ is a *saddle point* in T if $1 < i < (n+1)/2$ and $T[i]$ is smaller than its parent but larger than both of its children.
Derive the expected number of saddle points in T.

(iii) $M(i,j)$ is a *saddle point* in M if $M(i,j)$ is smaller than all of its neighbors in the same row and larger than all of its neighbors in the same column.
Derive the expected number of saddle points in M.

Structure of the Solution: The solution to each of the six subproblems must be written separately, with space between the solution to each subproblem

For each subproblem, you should derive the expected number of local minima or saddle points. as follows, using a three part solution:

(A) First clearly define the indicator random variable(s) that you will be using to solve this problem.

(B) Next, derive the expectation of each such indicator random variable.

(C) Finally, solve the full subproblem, by deriving a closed formula in n (no summations "$\sum$" allowed) for the requested value.

Solution The analysis of all of the subproblems use variants of the same simple well known fact. We write the most general form here for use.

Lemma 1: Let $A[1 \ldots n]$ form a uniform random permutation.
Let $Z = (i_1, i_2, \ldots, i_k) \subset \{1 \ldots n\}$. Then

$$\Pr\Big(A[i_1] > A[i_2] > \cdots > A[i_k]\Big) = \frac{1}{k!}.$$

Intuitively, this just says that in a random permutation, the items in any specified subset of the indices also forms a random permutation. **It was not necessary to prove this, but we provide a proof below for completeness.**

Proof: We first introduce some standard notation. S_n denotes the set of $n!$ permutations on n items and $\pi \in S_n$ denotes a single permutation. $\pi[i]$ is the i'th item in the permutation π.

So, stating that A is a uniform random permutation means that we choose $\pi \in S_n$ uniformly at random and set $A[i] = \pi[i]$ for every i.

Now let Q be the set of all permutations π such that $\pi[i_1] > \pi[i_2] > \cdots > \pi[i_k]$. We want the probability that we chose a $\pi \in Q$, i.e., $|Q|/|S_n|$.

For any $\pi \in Q$, let P_π be the set of all permutations $\pi' \in S_n$ such that

- $\pi'[i] = \pi[i]$ if $i \notin \{i_1, i_2, \ldots, i_k\}$
- $\pi'[i] \in \{\pi[i_1], \pi[i_2] \ldots, \pi[i_k]\}$ if $i \in \{i_1, \ldots, i_k\}$

That is, π' is identical to π outside of the designated indices but the entries in the designated indices can be permuted.

Now note that

(i) For every $\pi \in Q$, $|P_\pi| = k!$.

(ii) Every $\pi' \in S_n$ lies in exactly one set P_π.

(iii) $\Rightarrow$ $|Q|k! = |S_n|$.

But then

$$\Pr\Big(A[i_1] > A[i_2] > \cdots > A[i_k]\Big) = \frac{|Q|}{|S_n|} = \frac{1}{k!}.$$

■

All 4 subproblems in part (a) use the same following corollary of Lemma 1.

Corollary 1: Let $A[1 \ldots n]$ form a uniform random permutation. Let $Z = \{i_1, i_2, \ldots, i_k) \subset \{1 \ldots n\}$. Then

$$\Pr\Big(A[i_1] > A[i_j] \text{ for all } j,\ 2 \le j \le k\Big) = \frac{1}{k} \Pr\Big(A[i_1] < A[i_j] \text{ for all } j,\ 2 \le j \le k\Big).$$

This just says that if you fix k locations, the probability that $A[i_1]$ is the largest. or smallest, value in the k locations is $1/k$.

Proof: Just sum over all the $(k-1)!$ permutations of $i_2, \ldots, i_k)$ to find that

$$\Pr\Big(A[i_1] > A[i_j] \text{ for all } j,\ 2 \le j \le k\Big) = \frac{(k-1)!}{k!} = \frac{1}{k}$$

and

$$\Pr\Big(A[i_1] < A[i_j] \text{ for all } j,\ 2 \le j \le k\Big) = \frac{(k-1)!}{k!} = \frac{1}{k}.$$

■

(ai) We solve this problem using the indicator random variable technique.

(A) Let

$$X_i = \begin{cases} 1 & \text{if node } A[i] \text{ is a local mimima} \\ 0 & \text{if node } A[i] \text{ is not a local mimima} \end{cases}, \quad X = \sum_{i=1}^{n} X_i.$$

(B) For the three cases of i, we have:

(1) $1 < i < n : A[i]$ has two neighbors $i-1, i+1$.
Set $(i_1, i_2, i_3) = (i, i-1, i+1)$. From Corollary 1,

$$\Pr(X_i = 1) = \Pr(A[i] < A[i-1] \text{ and } A[i] < A[i+1]) = \frac{1}{3}.$$

(2) $i = 1 : A[i]$ has one neighbor $A[2]$.
Set $(i_1, i_2) = (1, 2)$. From Corollary 1,

$$E(X_1) = \Pr(X_1 = 1) = \Pr(A[1] < A[2]) = \frac{1}{2}.$$

(3) $i = n$. $A[n]$ has one neighbor $A[n-1]$.
Set $(i_1, i_2) = (n, n-1)$. From Corollary 1,

$$E(X_n) = \Pr(X_n = 1) = \Pr(A[n] < A[n-1]) = \frac{1}{2}.$$

(C) Combining pieces and using linearity of expectation gives

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = 1 \cdot \frac{1}{2} + (n-2) \cdot \frac{1}{3} + 1 \cdot \frac{1}{2} = \frac{n+1}{3}.$$

(aii) (A) Let

$$X_i = \begin{cases} 1 & \text{if node } T(i) \text{ is a local minima} \\ 0 & \text{if node } T(i) \text{ is not a local minima} \end{cases}, \quad X = \sum_{i=1}^{n} X_i.$$

(B) Recall that $n = 2^k - 1$ so T is a full binary tree of height $k-1$.
As illustrated in the figure, the tree T contains three types of nodes:

(1) 1 root node with no parent and two children;
(2) $2^k - 1 - 1 - 2^{k-1} = 2^{k-1} - 2 = \frac{n-3}{2}$ internal nodes; each has a parent and two children; and
(3) $2^{k-1} = \frac{n+1}{2}$ leaf nodes with a parent but no children.

We examine the three cases separately.

(1) Root Node $T[1]$: has two children $T[2], T[3]$ and is a local minimum if and only if $T[1] < T[2]$ and $T[1] < T[3]$. From Corollary 1,

$$E(X_1) = \Pr(X_1 = 1) = \Pr\Big(T[1] < T[2] \text{ and } T[1] < T[3]\Big) = \frac{1}{3}.$$

(2) Internal Nodes: Internal node $T[i]$ has three neighbors $T[\lfloor i/2 \rfloor \rfloor], T[2i], T[2i+1]$. From Corollary 1,

$$E(X_1) = \Pr(X_1 = 1) = \Pr\Big(T[i] < T[\lfloor i/2 \rfloor \rfloor, T[i] < T[2i], \, T[i] < T[2i{+}1]\Big) = \frac{1}{4}.$$

(3) Leaf Nodes: A leaf node $T[i]$ has only the one neighbor $T[\lfloor i/2 \rfloor]$ (its parent). From Corollary 1,

$$E(X_1) = \Pr(X_1 = 1) = \Pr\Big(T[i] < T[\lfloor i/2 \rfloor \rfloor\Big) = \frac{1}{2}.$$

(C) Combining pieces and using linearity of expectation gives

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = 1{\cdot}\frac{1}{3} + (2^{k-1} - 2){\cdot}\frac{1}{4} + (2^{k-1}){\cdot}\frac{1}{2} = \frac{3}{4}{\cdot}2^{k-1} - \frac{1}{6},$$

or, equivalently,

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = 1{\cdot}\frac{1}{3} + \left(\frac{n-3}{2}\right){\cdot}\frac{1}{4} + \left(\frac{n+1}{2}\right){\cdot}\frac{1}{2} = \frac{9n+5}{24}.$$

(aiii) (A) Let

$$X_{i,j} = \begin{cases} 1 & \text{if node } M(i,j) \text{ is a local minima} \\ 0 & \text{if node } M(i,j) \text{ is not a local minima} \end{cases}, \quad X = \sum_{i=1}^{m} \sum_{j=1}^{m} X_{i,j}.$$

(B) There are three types of entries:

(a) 4 corner entries $M(i,j)$:
These are the entries in which $i \in \{1, m\}$ and $j \in \{1, m\}$.
Each of these corner entries has two neighbors so, again from Corollary 1, for these 4 nodes $E(X_{i,j}) = \frac{1}{3}$.

(b) $4(m-2)$ non-corner edge entries $M(i,j)$.
These are the ones in which
$i \in \{1, m\}$ and $1 < j < m$ or $j \in \{1, m\}$ and $1 < i < m$.
Each of these entries has three neighbors so, again from Corollary 1, for these entries $E(X_{i,j}) = \frac{1}{4}$.

(c) $(m-2)^2$ internal entries:
These are the ones in which $1 < i < m, 1 < j < m$.
Each of these entries has four neighbors so, again from Corollary 1, for these entries $E(X_{i,j}) = \frac{1}{5}$.

(C) Combining pieces and using linearity of expectation gives

$$E(X) = \sum_{i=1}^{m} \sum_{j=1}^{m} E(X_{i,j}) = 4 \cdot \frac{1}{3} + 4(m-2) \cdot \frac{1}{4} + (m-2)^2 \cdot \frac{1}{5} = \frac{3m^2 + 3m + 2}{15}$$

or, equivalently,

$$E(X) = \frac{3n + 3\sqrt{n} + 2}{15}.$$

(bi) (A) Let

$$X_i = \begin{cases} 1 & \text{if node } A(i) \text{ is a saddle point} \\ 0 & \text{if node } A(i) \text{ is not a saddle point} \end{cases}, \quad X = \sum_{i=1}^{n} X_i.$$

(B) $A[i]$is a saddle point if and only if $1 < i < n$ and

$$A[i-1] < A[i] < A[i+1] \quad \text{or} \quad A[i-1] > A[i] > A[i+1].$$

From Lemma 1,

$$\Pr(A[i-1] < A[i] < A[i+1]) = \frac{1}{6} = \Pr(A[i-1] > A[i] > A[i+1]).$$

(C) Finally,

$$E(X_i) = \Pr(X_i = 1) = 2 \cdot \frac{1}{6} = \frac{1}{3}.$$

and

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = (n-2) \cdot \frac{1}{3} = \frac{n-2}{3}$$

(bii) (A) Let

$$X_i = \begin{cases} 1 & \text{if node } T(i) \text{ is a saddle point} \\ 0 & \text{if node } T(i) \text{ is not a saddle point} \end{cases}, \quad X = \sum_{i=1}^{n} X_i.$$

(B) First note that the condition $1 < i < (n+1)/2$, is equivalent to $T(i)$ being an internal node as described in (aiii).

$T[i]$ is a saddle point if and only if it is smaller than its parent and larger than its two children. This is equivalent to exactly one of the following two conditions occurring:

$$T[\lfloor i/2 \rfloor] > T[i] > T[2i] > T[2i{+}1] \quad \text{or} \quad T[\lfloor i/2 \rfloor] > T[i] > T[2i{+}1] > T[2i].$$

From Lemma 1, each of those two events has probability $\frac{1}{4!} = \frac{1}{24}$. Thus

$$E(X_i) = \Pr(X_i = 1) = \frac{2}{24} = \frac{1}{12}.$$

(C) For $i = 1$ or $i \geq (n+1)/2$, $E(X_i) = \Pr(X_i = 1) = 0$.

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = (2^{k-1} - 2) \cdot \frac{1}{12} = \frac{2^{k-2} - 1}{6}$$

or equivalently,

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = (\frac{n-3}{2}) \cdot \frac{1}{12} = \frac{n-3}{24}.$$

(biii) (A) Let

$$X_{i,j} = \begin{cases} 1 & \text{if entry } M(i,j) \text{ is a saddle point} \\ 0 & \text{if entry } M(i,j) \text{ is not a saddle point} \end{cases}, \quad X = \sum_{i=1}^{m} \sum_{j=1}^{m} X_{i,j}.$$

(B) Recall the three different types of node categorized in (aiv).

(a) Four corner entries.
We only examine the corner node $M(1,1)$. The others have the same analysis by symmetry.
$M(1,1)$ is a saddle point if and only if

$$M(2,1) < M(1,1) < M(1,2).$$

From Lemma 1,

$$E(X_{i,j}) = \Pr(X_{i,j} = 1) = \frac{1}{3!} = \frac{1}{6}.$$

By a symmetric argument, each of the other 3 corner nodes also has expected indicator RV value $\frac{1}{6}$.

(b) $4(m-2)$ non corner edge entries.
We only examine the edge where $i = 1$ and $1 < j < m$. The others have the same analysis by symmetry.
$M(1,j)$ is a saddle point if and only if

$$M(1,j-1) > M(1,j+1) > M(1,j) > M(2,j)$$

or

$$M(1,j+1) > M(1,j-1) > M(1,j) > M(2,j).$$

From Lemma 1, each of those two events has probability $\frac{1}{4!} = \frac{1}{24}$. Thus

$$E(X_{1,j}) = \Pr(X_{1,j} = 1) = \frac{2}{24} = \frac{1}{12}.$$

By a symmetric argument, every other edge node has expected indicator RV value $\frac{1}{12}$.

(c) $(m-2)^2$ internal entries.
$M(i,j)$ is a saddle point if and only if one of the four following conditions occurs:

$$M(i-1,j) > M(i+1,j) > M(i,j) > M(i,j-1) > M(i,j+1)$$

or

$$M(i+1,j) > M(i-1,j) > M(i,j) > M(i,j-1) > M(i,j+1)$$

or

$$M(i-1,j) > M(i+1,j) > M(i,j) > M(i,j+1) > M(i,j-1)$$

or

$$M(i+1,j) > M(i-1,j) > M(i,j) > M(i,j+1) > M(i,j-1)$$

From Lemma 1, each of those two events has probability $\frac{1}{5!} = \frac{1}{120}$. Thus

$$E(X_{i,j}) = \Pr(X_{i,j} = 1) = \frac{4}{120} = \frac{1}{30}.$$

(C) Combining pieces and using linearity of expectation gives

$$\begin{aligned} E(X) = E\left(\sum_{i=1}^{n} X_i\right) &= \sum_{i=1}^{n} E(X_i) \\ &= 4 \cdot \frac{1}{6} + 4(m-2) \cdot \frac{1}{12} + (m-2)^2 \cdot \frac{1}{30} \\ &= \frac{m^2 + 6m + 4}{30} \end{aligned}$$

or, equivalently,

$$E\left(\sum_{i=1}^{n} X_i\right) = \frac{n + 6\sqrt{n} + 4}{30}$$

Marking Note: By definition, the total number of local minima or saddle points is at most the number of items, i.e., n. It is therefore impossible that $E(\sum_{i=1}^{n} X_i) > n$. Solutions that stated this result, e.g., by reporting the answer as $\Theta(n^2)$, were therefore totally incorrect.

The instructions were specific that the solution had to be in closed form and not summations of $\Theta(n)$ terms. Solutions not in closed form had points deducted.

Also, a correct solution required understanding that there are multiple different types of nodes and each one requires a slightly different analysis.

Problem 3 [24 pts] (Using Selection)

Note: Recall the Selection problem of calculating Select(A, p, r, i) that we learned in class. We actually learned a randomized *linear, i.e., $O(r - p + 1)$, time algorithm for solving Selection. We also stated that a* deterministic *linear time algorithm for solving this problem exists. For the sake of this problem, you may assume that you have been given this deterministic linear time algorithm and can use it as a subroutine. (calling it exactly as Select(A, p, r, i)). You may not use any other algorithm as a subroutine without explicitly providing the code and proving correctness from scratch.*

The *Manhattan Distance* between two 2-dimensional points $p = (x, y)$ and $p' = (x', y')$ is

$$d_1(p, p') = |x - x'| + |y - y'|.$$

In what follows you are given real numbers $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. They are NOT necessarily sorted.

Define the n two-dimensional points $p_i = (x_i, y_i)$. The *Manhattan Warehouse problem* is to find a point $p = (x, y)$ that minimizes the average distance to all the p_i. That is, to find p such that

$$\forall p' = (x', y') \in \Re^2, \quad \sum_{i=1}^{n} d_1(p, p_i) \leq \sum_{i=1}^{n} d_1(p', p_i).$$

(Dividing both sides of the equation by $\frac{1}{n}$ gives the average.)

The goal of this problem is to design a $O(n)$ worst case time algorithm for solving the Manhattan Warehouse problem. We split this into two parts.

(A) Solve the one-dimensional problem.
Given (unsorted) $x_1, \ldots x_n$, define the function

$$f(x) = \sum_{i=1}^{n} |x - x_i|.$$

Call $\bar{x}$ a *center* if it minimizes $f(x)$, i.e.,

$$\forall x \in \Re, \quad f(\bar{x}) \leq f(x).$$

(a) Prove that there exist z_1, z_2 such that $z_1 \leq z_2$ and
(i) $f(x)$ is monotonically decreasing for $x \in [-\infty, z_1]$.
(ii) $f(z_1) = f(x) = f(z_2)$ for all $x \in [z_1, z_2]$.
(iii) $f(x)$ is monotonically increasing for $x \in [z_2, \infty]$.
Note that it is possible that $z_1 = z_2$.

(b) Give an $O(n)$ time algorithm for finding a center of n one-dimensional points $x_1, \ldots, x_n$.
First give documented pseudocode for your algorithm.
In addition, below your algorithm, explain in words/symbols what your algorithm is doing

(c) Prove correctness of your algorithm. Your proof must be mathematically formal and understandable.

(d) Explain why your algorithm runs in $O(n)$ time.

(B) Solve the Manhattan Warehouse Problem.

(a) Give an $O(n)$ time algorithm for solving the Manhattan Warehouse Problem given n points $p_1, p_2, \ldots, p_n$ where $p_i = (x_i, y_i)$.
First give documented pseudocode for your algorithm.
In addition, below your algorithm, explain in words/symbols what your algorithm is doing

(b) Prove correctness of your algorithm. Your proof must be mathematically formal and understandable.

(c) Explain why your algorithm runs in $O(n)$ time.

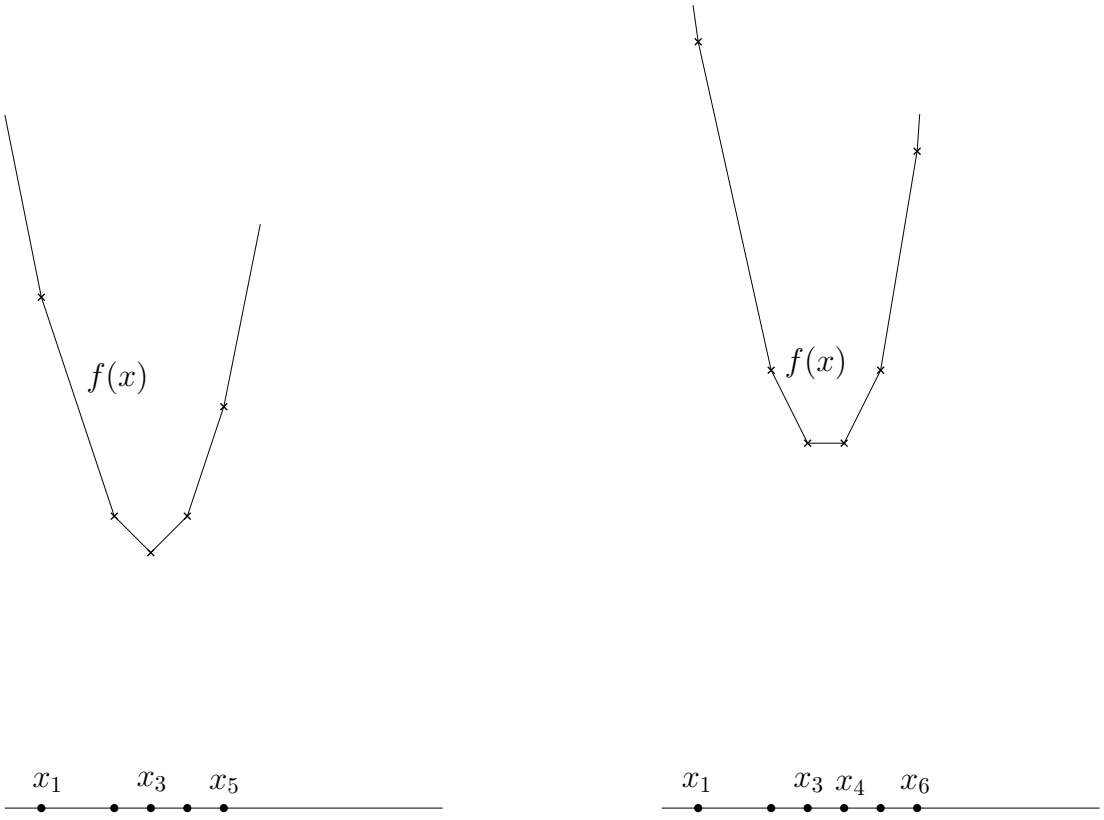
Notes:

- *For simplicity, you may assume that none of the x_i or y_i repeat, i.e., there is no pair i, j such that $x_i = x_j$ or $y_i = y_j$.*
- *Part A(a) is only meant as a hint to get you started. Note that if you know z_1, z_2, then (you must prove this) any $x \in [z_1, z_2]$ will be a solution to the one-dimensional center problem.*

  A more detailed understanding of the behavior of $f(x)$, that lets you find z_1, z_2, therefore permits solving the remainder of A. You may, if you like, fully analyze the behavior of $f(x)$ in A(a), write this as a mathematical lemma and then quote that lemma in A(c).
- *We strongly recommend that before trying to solve part A you choose 5 or 6 points x_i and then graph the resultant function $f(x)$. Seeing the diagram should help you solve the problem.*
- *The algorithm for part (B) can use the result of part (A) as a subroutine.*
- *The main work in solving this problem is in understanding how to solve it using selection. Once you understand that, the actual pseudocode might be quite short.*

Solution:

Before starting it is worthwhile to look at examples of $f(x)$ drawn for 5 and 6 points. For 5 points the minimum is exactly at $f(x_3)$. For 6 points, the minumum is any x satisfying $x_3 \leq x \leq x_4$.



Note that in the first problem, $f(x)$ reaches its minimum exactly at $x = x_3$, the median point. In the second, $f(x)$ reaches its minimum at all $x \in [x_3, x_4]$, i.e., at all points *between* the two minimum.

Caveat: To simplify the illustration, the example assumed that the x_i are sorted. In the real input they are not.

The main work in the problem was to identify that it can be reduced to finding the median which can be done in $O(n)$ time WITHOUT sorting the input.

(A) (a) For the analysis (NOT for the algorithm), sort the x_i and relabel them as $y_1, y_2, \ldots, y_n$ where $y_1 < y_2 < \cdots < y_n$. Then, the y_i are just a relabelling of the x_i, so

$$f(x) = \sum_{i=1}^{n} |x - x_i| = \sum_{t=1}^{n} |x - y_t|.$$

The main fact is

Lemma:

$$f(x) = \begin{cases} -nx & + \Big(f(y_1) + ny_1\Big) & \text{if } y \le y_1 \\ -(n-2i)x & + \Big(f(y_i) + (n-2i)y_i\Big) & \text{if } 1 \le i < n \text{ and } y_i \le y \le y_{i+1} \\ nx & + \Big(f(y_n) - ny_n\Big) & \text{if } y \ge y_n \end{cases}$$

Proof:

Note that if $x \le y_1$ then x is to the left of all of the points so

$$f(x) = \sum_{t=1}^{n} (y_t - x) = \sum_{t=1}^{n} ((y_t - y_1) + (y_1 - x)) = -nx + ny_1 + f(y_1).$$

Similarly, if $y \ge y_n$ then x is to the right of all of the points so

$$f(x) = \sum_{t=1}^{n} (x - y_t) = \sum_{t=1}^{n} ((x - y_n) + (y_n - y_t)) = nx - ny_n + f(y_n).$$

Now, consider $x \in [y_i,\, y_{i+1}]$. Then

$$\begin{aligned} f(x) &= \sum_{t=1}^{n} |x - y_t| \\ &= \sum_{t=1}^{i} (x - y_t) + \sum_{t=i+1}^{n} (y_t - x) \\ &= \sum_{t=1}^{i} (x - y_i + y_i - y_t) + \sum_{t=i+1}^{n} (y_t - y_i + y_i - x) \\ &= \sum_{t=1}^{i} (x - y_i) + \sum_{t=1}^{i} (y_i - y_t) + \sum_{t=i+1}^{n} (y_t - y_i) + \sum_{t=i+1}^{n} (y_i - x) \\ &= \left(\sum_{t=1}^{i} (x - y_i) + \sum_{t=i+1}^{n} (y_i - x)\right) + \left(\sum_{t=1}^{i} (y_i - y_t) + \sum_{t=i+1}^{n} (y_t - y_i)\right) \\ &= -(n-2i)x + (n-2i)y_i + f(y_i). \end{aligned}$$

■

Recall that $y = mx + b$ is the equation of a line.

- If $m < 0$, y is a decreasing function;
- if $m = 0$, y is a constant function;
- and if $m > 0$, y is an increasing function.

The lemma states that $f(x)$ is a piecewise linear function with slope $-n$ towards $-\infty$, slope n towards ∞ and slope that is increasing by 2 at each successive y_i value.

The slope of $f(x)$ is negative as long as $n - 2i > 0$ and the slope is positive when $n - 2i < 0$.

Note that that if n is even, for $i = n/2$, $n - 2i = 0$ so there's an interval in which the slope is 0.

If n is odd then $n - 2i > 0$ for $i < \lceil n/2 \rceil$ and $n - 2i < 0$ for $i \geq \lceil n/2 \rceil$

This indicates that we need to split the analysis into two cases. One for n even and one for n odd.

If n is odd: the slope of $f(x)$ switches from negative to positive at $y_{\lceil n/2 \rceil}$ so

- $f(x)$ is monotonically decreasing for $x \leq y_{\lceil n/2 \rceil}$ and
- monotonically increasing for $x > y_{\lceil n/2 \rceil}$.

The statement is therefore correct for $z_1 = y_{\lceil n/2 \rceil} = z_2$.
Note that this implies that $f(x)$ achieves its minimum at $x_0 = y_{\lceil n/2 \rceil}$.

If n is even:

- the slope of $f(x)$ is negative for $x < y_{n/2}$ and
- positive for $x < y_{n/2+1}$.
- The slope of $f(x)$ is zero for $y_{n/2} \leq x \leq y_{n/2+1}$.

The statement is therefore correct for $z_1 = y_{n/2}$ and $z_2 = y_{n/2+1}$.

Note that this implies that $f(x)$ achieves its minimum for every x satisfying $y_{n/2} \leq x \leq y_{n/2+1}$.

(b) The algorithm is

$Center(x_1, \ldots, x_n)$:
1. Create array $A[1 \ldots n]$ with $A[i] = x_i$. % Create Array A storing the x_i
2. $Return(Select(A, 1, n, \lceil n/2 \rceil))$. % Find median of the x_i

This is just using the given procedure to find the *median* item, i.e., the $\lceil n/2 \rceil$'th one.

(c) Correctness follows directly from (a)

(d) Follows from the linearity of the *Select* procedure given at the start of the problem.

(B)

(a) $Warehouse(x_1, \ldots, x_n, y_1, \ldots, y_n)$:
1. $x = Center(x_1, \ldots, x_n)$ % Find median of the x_i
2. $y = Center(y_1, \ldots, y_n)$ % Find median of the y_i
3. $Return((x, y))$

Separately find the medians of the x_i and y_i. Combined, those give you the x and y coordinates of the solution to the Warehouse problem.

(b) We are trying to find $p = (x, y) \in \Re^2$ that minimizes

$$\sum_{i=1}^{n} d_1(p, p_i) = \left(\sum_{i=1}^{n} |x - x_i| \right) + \left(\sum_{i=1}^{n} |y - y_i| \right).$$

Since there is no connection between the x part and the y part we can minimize this by separately finding x that minimizes $\sum_{i=1}^{n} |x - x_i|$ and y that minimizes $\sum_{i=1}^{n} |y - x_i|$. But we already saw how to do this in part (a).

(c) Follows from the linearity of the procedure given in part (a).

Marking Note: Any solution for parts (A) or (B) that did not use selection was wrong.

More explicitly, (A) was equivalent to finding the median and (B) was equivalent to separately finding the median of the x_i and y_i.

Any solution that did not use the selection algorithm provided was wrong.

More specifically, any $O(n)$ solution to either (A) or (B) would give an $O(n)$ time solution for finding the median which would also (this is known) give an $O(n)$ time algorithm for solving selection.

So, if your solution did not USE the selection procedure provided it had to have created a new $O(n)$ time one, which no one did properly ($O(n)$ time selection is sophisticated and complicated, which is why we provided the procedure. You weren't expected to create one).

Also, some students just said "find the median" without explaining HOW. You needed to explicitly explain that the selection algorithm when called with the correct parameters, returns the median.

P4: [18 pts] **Recurrence Relations**

Give asymptotic upper bounds for $T(n)$ satisfying the following recurrences. Make your bounds as tight as possible. For example, if $T(n) = \Theta(n^2)$ then $T(n) = O(n^2)$ is a tight upper bound but $T(n) = O(n^2 \log n)$ is not. Your upper bound should be written in the form $T(n) = O(n^\alpha (\log n)^\beta)$ where α, β are appropriate constants.

A correct answer will gain full credits. It is not necessary to show your work BUT, if your answer was wrong, showing your work steps may gain you partial credits.

If showing your work, you may quote theorems shown in class.
For (a), (b) and (c) you may assume that n is a power of 2;
for (d) you may assume that n is a power of 4;
for (e) that n is a power of 7; for (f) that n is a power of 3

(a) $T(1) = 1; T(n) = 4T(n/2) + n^2\sqrt{n}$ for $n > 1$.

(b) $T(1) = 1; T(n) = 9T(n/2) + n^3 \log_2 n$ for $n > 1$.

(c) $T(1) = 1; T(n) = 4T(n/2) + \sum_{i=1}^{n} i$ for $n > 1$.

(d) $T(1) = 1; T(n) = 3T(n/4) + 1$ for $n > 1$.

(e) $T(1) = 1; T(n) = 2T(n/7) + 2^{\log_7 n}$ for $n > 1$.

(f) $T(1) = 1; T(n) = 9T(n/3) + \log_3 (n!)$ for $n > 1$.

Solution:

(a) $T(n) = O(n^{5/2})$; *(Case 3 in Master Theorem)*

(b) $T(n) = O(n^{\log_2 9})$; *(Case 1 in Master Theorem)*

(c) $T(n) = O(n^2 \log n)$; *(Case 2 in Master Theorem)*

(d) $T(n) = O(n^{\log_4 3})$; *(Case 1 in Master Theorem)*

(e) $T(n) = O(n^{\log_7 2} \log n)$; *(Case 2 in Master Theorem)*

(f) $T(n) = O(n^2)$; *(Case 1 in Master Theorem)*

Comments:

(a) $f(n) = n^{5/2}$, $c = \log_2 4 = 2 < 5/2$.
Note that the smoothness condition $af(n/b) \leq df(n)$ *for some* $d < 1$ *is satisfied if and only if for some* $d < 1$

$$4\left(\frac{n}{2}\right)^{5/2} \leq dn^{5/2} \quad \Leftrightarrow \quad 4\left(\frac{1}{2}\right)^{5/2} \leq d \quad \Leftrightarrow \quad \left(\frac{1}{2}\right)^{1/2} \leq d.$$

So, setting $d = \frac{1}{\sqrt{2}}$ *shows that the smoothness condition is satisfied.*

(b) $f(n) = n^3 \log_2 n$, $c = \log_2 8 > 3$.

(c) $f(n) = \Theta(n^2)$, $c = \log_2 4 = 2$.

(d) $f(n) = 1 = n^0$, $c = \log_4 3 > 0$.

(e) $f(n) = n^{\log_7 2}$, $c = \log_7 2$.

(f) $f(n) = \log_3(n!) = \Theta(n \log n)$, $c = \log_3 9 = 2$.

Marking Note: there is no need to specify the master theorem case. Partial scores could be given to correct steps if shown (even if the final answer was wrong).