CSE 60111: Complexity and Algorithms
Homework 1

1. (10 points) Solve the following recurrences, giving tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$, or at least upper and lower bounds (big-O and big-Ω). Assume reasonable but nontrivial base cases if none are supplied. More exact solutions are better.

   (a) $A(n) = A(n/2) + n^2$

   (b) $B(n) = B(n/2) + \lg n$

   (c) $C(n) = 3C(n/2) + n$

   (d) $D(n) = \max_{1 \le k \le n/2}(D(k) + D(n-k) + n)$

   (e) $E(n) = E(3n/5) + E(n/5) + n^2$

2. (10 points) Suppose we are given a set S of n items, each with a value and a weight. For any element $x \in S$, we define two subsets:

   - $S_{<x}$ is the set of elements of S whose value is less than the value of x.
   - $S_{>x}$ is the set of elements of S whose value is more than the value of x.

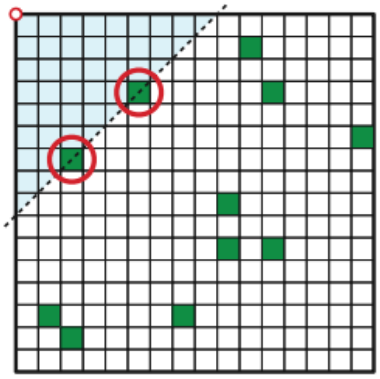
   For any subset $R \subseteq S$, let $w(R)$ denote the sum of the weights of elements in R. The weighted median of R is any element x such that $w(S_{<x}) \le w(S)/2$ and $w(S_{>x}) \le w(S)/2$.

   Describe and analyze an algorithm to compute the weighted median of a given weighted set as quickly as possible. Your input consists of two unsorted arrays $S[1..n]$ and $W[1..n]$, where for each index i, the i^{th} element has value $S[i]$ and weight $W[i]$. You may assume that all values are distinct and all weights are positive.

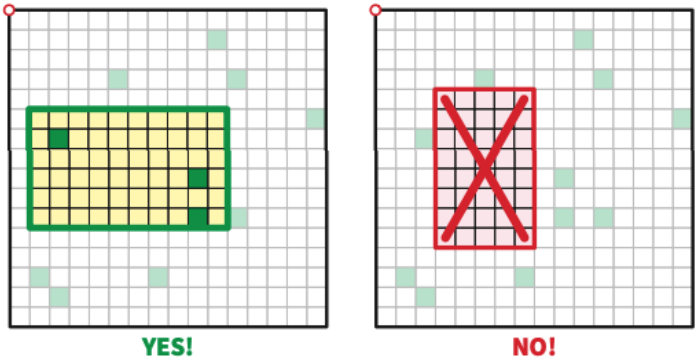
   (Hint: You can use/adapt the median of medians here!)

3. (15 points) Before his untimely death, Captain Jack Sparrow buried several bottles of rum on Grid Island. You have a map of Rum Island in the form of an $n \times n$ grid of squares, with rows indexed from north to south and columns from west to east; each square on the map may or may not correspond to the location of a bottle of rum. The only safe port on Grid Island is in the northwest corner, next to square $(1, 1)$. You want to find a bottle of rum that is as close as possible to the port; specifically, you want to find a grid square (i, j) that contains a bottle of rum, such that $i + j$ is as small as possible.

   For example, in the figure below, solid green squares represent bottles of rum; your ship is docked in the northwest corner. Your goal is to find one of the circled bottles of rum:



   To aid in your quest, Davy Jones has given you a magical Device that will correctly report whether any rectangular area on Grid Island contains at least one bottle of rum. That is, given four integers t, b, l, r between 1 and n, if there is a rum bottle in any square (i, j) such that $t \leq i \leq b$ and $l \leq j \leq r$, the Device sounds a gentle mellifluous chime; if there is no rum in that rectangular region, the Device emits a fart noise and a noxious cloud of green "smoke". Each use of the Device takes $O(1)$ time.



   You could use the Device to search the grid one square at a time, but that would require $\Theta(n^2)$ time in the worst case. Surely we can do better!

   (a) Consider the following recursive algorithm to find a bottle of rum closest to the northwest corner of the island.
   - If $n = 1$, the only square must contain the only bottle of rum.
   - Otherwise, split the $n \times n$ into four $n/2 \times n/2$ quadrants, use the Device to test the northwest quadrant. If the northwest quadrant does not contain any rum, then it certainly does not contain the closest rum to your ship. Recursively search the northeast, southwest, and southeast quadrants.

- On the other hand, if the northwest quadrant does contain rum, then the closest rum to your ship cannot be in the southeast quadrant. Recursively search the northwest, northeast, and southwest quadrants.

In both cases, each of the three recursive calls either returns the location of the closest rum to your ship in that quadrant or correctly reports that there is no rum in that quadrant. Moreover, at least one of the recursive calls finds some rum. The closest rum to your ship is the closest of the one, two, or three bottles found by the recursive calls. What is the overall running time of this recursive algorithm?

(b) Now suppose we modify the algorithm in part (a) to partition into nine regions instead of four. At each level of recursion, the new algorithm splits Grid Island into nine $(n/3) \times (n/3)$ subgrids, tests all nine subgrids using the Device, and then recursively searches a subset of those nine subgrids.

How many recursive calls do we need to make in the worst case? Prove your answer is correct. What is the running time of the resulting recursive algorithm?

(c) Describe an even faster algorithm to find a bottle of rum that is closest to the northwest corner of Grid Island. (Note that slower correct algorithms that are still faster than parts (a) and (b) will receive significant partial credit, even if you don't get the best possible run time in your solution.)

4. (10 points) You are a visitor at a political convention (or perhaps a faculty meeting) with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the same party by introducing them to each other. Members of the same political party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

(a) Suppose more than half of the delegates belong to the same political party. Describe an efficient algorithm that identifies all members of this majority party.

(b) Now suppose there are more than two parties, but one party has a plurality: more people belong to that party than to any other party. Present a practical procedure to precisely pick the people from the plurality political party as parsimoniously as possible, presuming the plurality party is composed of at least p people. Pretty please.

5. Sample solved problem:

   You are interested in analyzing some hard-to-obtain data from two separate databases, which I'll call A and B. Each database contains n numerical values, so that there are $2n$ total, and you may assume that no two are the same. You'd like to determine the median value of this set of $2n$ values, which we define to be the n^{th} value.

   However, the situation is complicated by the fact that you can only access these values through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k^{th} smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

   Given an algorithm that finds the median value using at most $O(\log n)$ queries. Be sure to specify the algorithm, the analysis for the number of queries, and a justification (i.e. a proof) that your algorithm returns the median value. (To keep things simpler, you are welcome to assume that n is a power of 2.)

   Solution:

   I'll index the elements of each database from 1 to n, so that query(A, i) looks up the i^{th} entry in the database A. Suppose we first query each database for its median—that is, we query each one for element $n/2$. Let m_1 be the median from database 1 and let m_2 be the median from database 2. Suppose, without loss of generality, that $m_1 < m_2$. We know that $n/2$ elements in database 1 are less than or equal to m_1, and therefore must also be less than m_2. (We don't yet know how the remaining $n/2$ elements in database 1 are ordered with respect to m_2). Similarly, we know that $n/2$ elements in database 2 are greater than m_2 and therefore also greater than m_1 (but we don't know how the remaining $n/2$ elements in database 2 are ordered with respect to m_1.

   At this point, we can conclude something: the median of all $2n$ elements must be in the largest half of database 1, or in the smallest half of database 2. At this point, we can chop of databases in half, and recurse!

   Pseudocode:

```
FindMedian(A, a_min, a_max, B, b_min, b_max):
  if a_max == a_min
      m_a ← query(A, 1)
      m_b ← query(B, q)
      return min(m_a, m_b)
  m_a ← query(A, (a_max + a_min − 1)/2)
  m_b ← query(B, (b_max + b_min − 1)/2)
  if m_a < m_b
      return FindMedian(A, m_a + 1, a_max, B, b_min, m_b)
  else
      return FindMedian(A, a_min, m_a, B, m_b + 1, b_max)
```

   Our initial call is then to FindMedian$(A, 1, n, B, 1, n)$; from there on out, $a_{\min}$ and $a_{\max}$ (as well as the similar values in B) represent the subdatabase that we are working with in that recursive call.

   Proof of correctness: Induction on n:

   Base case: There is a single element in each database, so the median is (by definition) the smaller of the two. This is done in the first if statement of our pseudocode.

Inductive Hypothesis: For any value $< n$ of items in each database, our algorithm correctly returns the median of all elements in the two databases.

Inductive Step: Consider a database of $2n$ elements (so exactly n in each individual one). We first find the two medians, m_a and m_2. As described above, if we have $m_a < m_b$, then elements in the first half of database A are less than both medians, and elements in the second half of database B are greater than both. Since there are $n/2$ elements in each of these halves, we know that the overall median cannot be in those sections: if it were, we would have a contradiction, because there are $n+1$ elements greater than m_a, and $n+1$ elements less than m_b. Since the median is the n^{th} smallest, this is impossible.

Since we discard the same number of elements from A and B, the median overall will be equal to the median of the smaller databases of size n each. By our inductive hypothesis, our algorithm will correctly find the median of these smaller databases, and thus will return the overall correct value.

Runtime: Our base case makes two queries and a comparison, which is total time $O(1)$.

We then can construct the recurrence as follows: Our pseudocode makes 2 queries, then does a comparison, then calculates two additions and two divisions, before making a recursive call on a database which is half as big. The resulting recurrence is

$$T(n) = T(n/2) + O(1)$$

Plugging into Master theorem, we get $T(n) = \Theta(\log n)$.