Homework 4: Solution outlines by your TA

Problem 1[Exercise 8.4-4]

Strategy: Modify Bucket-Sort so that each bucket represents a radius reflecting the uniform distribution of the points in the circle.

We will divide the circle into n concentric rings, such that each ring has equal area. Since the total area of the circle is π, each ring must have area $\frac{\pi}{n}$.

Innermost ring: We must find r_1 such that the ring of all points whose distance from the center of the circle is between 0 and r_1 has area $\frac{\pi}{n}$. This ring is just a circle, however, and so we have
$\pi r_1^2 = \frac{\pi}{n}$, or $r_1^2 = \frac{1}{n}$, or $r_1 = \frac{1}{\sqrt{n}}$. So, the first bucket will hold points whose distances from the origin are between 0 and $\frac{1}{\sqrt{n}}$.

Second ring: We must find r_2 such that the ring of all points whose distance from the center of the circle is between r_1 and r_2 has area $\frac{\pi}{n}$. But we note that this ring together with the innermost ring forms a circle of radius r_2. We know that the innermost ring (circle) has area $\frac{\pi}{n}$, and so this composite circle must have area $\frac{2\pi}{n}$. Thus, we must find r_2 such that $\pi r_2^2 = \frac{2\pi}{n}$, or $r_2^2 = \frac{2}{n}$, or $r_2 = \frac{\sqrt{2}}{\sqrt{n}}$. So, the second bucket will hold points whose distances from the origin are between $\frac{1}{\sqrt{n}}$ and $\frac{\sqrt{2}}{\sqrt{n}}$.

Third ring: We must find r_3 such that the ring of all points whose distance from the center of the circle is between r_2 and r_3 has area $\frac{\pi}{n}$. But with the same argument as above (innermost 3 rings form a circle), this is equivalent to finding r_3 such that $\pi r_3^2 = \frac{3\pi}{n}$, or $r_3^2 = \frac{3}{n}$, or $r_3 = \frac{\sqrt{3}}{\sqrt{n}}$. So, the third bucket will hold points whose distances from the origin are between $\frac{\sqrt{2}}{\sqrt{n}}$ and $\frac{\sqrt{3}}{\sqrt{n}}$.

Continuing in this manner, we can see by a simple inductive argument that the ith bucket will hold points whose distance from the origin is greater than $\frac{\sqrt{i-1}}{\sqrt{n}}$ and less than or equal to $\frac{\sqrt{i}}{\sqrt{n}}$. Since it was shown above that each such bucket represents a part of the circle whose area is $\frac{1}{n}$ of the total area of the circle, and since the points are known to be uniformly distributed by area throughout the circle, these buckets are appropriate for the Bucket Sort algorithm. Since each point's distance can be computed in constant time from its x and y coordinates, the Bucket Sort algorithm used with these bucket sizes will sort any group of n points so distributed throughout the unit circle by their distances from the origin in $O(n)$ time, as required.

Problem 2[Exercise 9.3-8] Given two sorted lists of numbers $X[1..n]$ and $Y[1..n]$ we need to come up with a $O(\lg n)$-time algorithm to find the median of all the $2n$ elements.

The idea is illustrated in Figure .

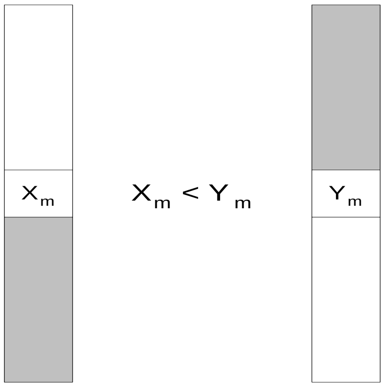

Figure 1: Median of two sorted lists lies in the shaded region

```
QuickMedian($X$, $i_x$, $j_x$, $Y$, $i_y$, $j_y$)
    1. $m_x = \lceil \frac{j_x - i_x}{2} \rceil$
       index of the median of the numbers $X[i_x..j_x]$
    2. $m_y = \lceil \frac{j_y - i_y}{2} \rceil$
       index of the median of the numbers $Y[i_y..j_y]$
    3. if $X[m_x] = Y[m_y]$
          return $X[x_m]$
    4. if $X[m_x] < Y[m_y]$
          return QuickMedian($X$, $m_x$, $j_x$, $Y$, $i_y$, $m_y$)
    5. else
          return QuickMedian($X$, $i_x$, $m_x$, $Y$, $m_y$, $j_y$)
```

We claim that QuickMedian(X, 1, n, Y, 1, n) will find the median of all the $2n$ elements correctly in $O(\lg n)$ time.

Correctness: We first find the medians of X and Y. If the two medians are equal, then it is the median of all the $2n$ elements, because, there are equal number of elements less than and greater than this element (line 3).

If the medians are not equal, we can have two possibilities: either median of X is less than median Y and the other way round. We consider the former case (the later can

be done by exchanging X and Y). ($X_m < Y_m$). Claim(1): Any element less than X_m in X cannot be the median.
Consider an element A less than X_m in X. There will be less than $n/2$ numbers less than A. Also since $A < X_m < Y_m$, there will be less than $n/2$ elements in Y that are less than A. Therefore there will be less than half the total elements less than A and hence, A can not be the median.

Similarly we can make the following claim.
Claim(2): Any element more than Y_m in Y cannot be the median.

From claims(1) & (2) we know that the median can lie only in the unshaded regions. These two unshaded arrays are still sorted and are of the same size, therefore we can recursively apply the algorithm to find the median.

Running time:
Lines 1, 2 take O(1) time. (median can be found in constant time since the lists are sorted). Line 3 is the termination of the recursion and takes constant time. Lines 4, 5 make a recursive call over input that is half the original size.

Therefore the running time $T(n) = T(n/2) + O(1) = O(\lg n)$.

Problem 3[Exercise 9.3-9]

Finding the optimal pipeline location

Firstly, since the pipes can only run vertically or horizontally, the east-west oil pipeline must run horizontally from the x-coordinate of the westmost well to the x-coordinate of the eastmost well. Therefore, a constant amount of east-west pipe must be used, and we are concerned mainly with the y-coordinate of the main pipeline, which will determine the amount of north-south pipe used.

The y-coordinate of the optimal east-west pipeline is at the median of the y-coordinates of the wells (or in the case of 2 medians, at any point between those). We divide the proof into two cases.

Case 1 n is odd

Consider an east-west pipeline whose y-coordinate is m, the median of the y-coordinates of all the wells (and the y-coordinate of some well w). Thus, $\frac{n-1}{2}$ of the wells are above the east-west pipeline, and $\frac{n-1}{2}$ are below. Now consider moving the east-west pipeline to y-coordinate $m' > m$. The total amount of north-south pipe is increasing by $m' - m$ from at least $\frac{n-1}{2}$ of the wells (those below the median well), but decreasing by $m' - m$ from only at most $\frac{n-1}{2}$ of the wells (those above the median well). Furthermore, the amount of north-south pipe connecting the median well to the main pipeline has increased by $m' - m$. Since $(m' - m) > 0$, the total amount of north-south pipe has increased, and so any y-coordinate of the main pipeline which is greater than m is non-optimal. The proof that any y-coordinate of the main pipeline which is less than m is non-optimal is completely similar.

Case 2 n is even

Consider an east-west pipeline whose y-coordinate is m, and let m be between m_1 and m_2, the two medians of the y-coordinates of all the wells. (Let $m_1 > m_2$.) Thus, $\frac{n-2}{2}$ of the wells are above the east-west pipeline, and $\frac{n-2}{2}$ are below. Now consider moving the east-west pipeline to y-coordinate $m' > m_1 \geq m$. The total amount of north-south pipe is increasing by $m' - m$ from at least $\frac{n-2}{2}$ of the wells (those below both median wells), but decreasing by $m' - m$ from only at most $\frac{n-2}{2}$ of the wells (those above both median wells). Furthermore, the total amount of north-south pipe connecting both of the median wells to the main pipeline has increased by $m' - m$. Since $(m' - m) > 0$, the total amount of north-south pipe has increased, and so any y-coordinate of the main pipeline which is above the higher of the two medians is non-optimal. The proof that any y-coordinate of the main pipeline which is below the lower of the two medians is non-optimal is completely similar.

Finally, to see that any location of the east-west pipeline whose y-coordinate is between the two medians (inclusive) has the same total amount of north-south pipe, consider a main pipeline located at the average of the y-coordinates of the two medians. If the main pipeline is moved to the north by distance d (but still remains below the higher median), then the total amount of north-south pipe connecting the two medians has remained the same, while the total amount of north-south pipe connecting the other wells to the main pipeline has increased by d for $\frac{n-1}{2}$ of the wells, and decreased by d for the other $\frac{n-2}{2}$ of the wells. Thus, the total amount of pipe has remained constant. Similarly, it will also remain constant if the main pipeline is moved to the south but still remains above the lower median.

Thus, any y-coordinate between the two medians (inclusive) is an optimal y-coordinate for the main east-west pipeline.

Problem 4 – Weighted Median

a. Argue that the median of $\{x_i\}$ is the weighted median of $\{x_i\}$ with all weights $w_i = \frac{1}{n}$.

Let x_k be the median of $\{x_1, x_2, ..., x_n\}$, where $x_i \neq x_j \forall i \neq j$, and let the $\{x_i\}$ have weights as above. By definition of median, at most half (or $\frac{n}{2}$) of the x_i are less than x_k and at most half are greater than x_k. Thus, the sum of the weights of elements less than x_k is at most $(\frac{n}{2})(\frac{1}{n})$, or $\frac{1}{2}$, and the sum of the weights of elements greater than x_k is at most $(\frac{n}{2})(\frac{1}{n})$, or $\frac{1}{2}$. Therefore, by definition, x_k is the weighted median of $\{x_i\}$, as required.

b. Show how to compute the weighted median of n elements in $O(n \lg n)$ worst-case time using sorting.

Sort-Weight-Median(X,n)

```
1. MergeSort(X,1,$n$) or HeapSort(X,1,$n$)
2. $sum \leftarrow 0$
3. $m \leftarrow 0$
4. while($sum < \frac{1}{2}$)
5.       $m \leftarrow m + 1$
6.       $sum \leftarrow sum + w_m$
7. end while
8. return $m$
```

This algorithm correctly finds the index m of the weighted median x_m, by using the definition of weighted median. It first sorts the data based on x's. Since, $sum_{x_i < x_m} < 1/2$, we add up the weights of the sorted x_i's until the sum is no longer less than 1/2. Also note that the sum of weights of the points left of x_m, i.e. $sum_{x_i > x_m} \leq 1/2$. Therefore, x_m is the weighted median.

This algorithm also runs in $O(n \lg n)$ time in the worst case:

Line 1: $\Theta(n \lg n)$
Lines 2,3: $\Theta(1)$
Lines 4,5,6: $(O(n))(\Theta(1))$, since the loop will iterate at most n times, and a constant amount of work is done for each iteration.
So, the total running time is $\Theta(n \lg n) + O(n) + \Theta(1) = \Theta(n \lg n) = O(n \lg n)$, as required.

c. Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm, such as SELECT from 9.3.

Linear-Weighted-Median(X,$start$,end)

```
1. $m \leftarrow \lceil \frac{end-start}{2} \rceil$
2. $X_m = SELECT(X, start, end, m)$
   finds the median of $X$ based on just the values and not the weights
3. $PARTITION(X, start, end, X_m)$
   partitions $X[start..end]$ using $X_m$ as the pivot.  elements less than $X_m$ to the left and others to the
   right of $X_m$
4. $Sum_{left} = \sum_{i=start}^{m-1} X_i$
   sum of weights of numbers less than the $X_m$
5. $Sum_{right} = \sum_{i=m+1}^{end} X_i$
   sum of weights of numbers more than the $X_m$
6. if $Sum_{left} < 1/2$ and $Sum_{right} \leq 1/2$
     return $X_m$
       this is the weighted median
7. if $Sum_{left} \geq 1/2$
     $X_m.weight \leftarrow X_m.weight + Sum_{right}$
       weight of the lighter side is added to the median
     return Linear-Weighted-Median($X$,$start$,$i$)
8. else
     $X_m.weight \leftarrow X_m.weight + Sum_{left}$
       weight of the lighter side is added to the median
     return Linear-Weighted-Median($X$,$i$,$end$)
```

We find the median(line 2). Then we partition the elements based on weather they are less than or more than the median (line 3). We find the sum of the weights of elements on both sides (lines 4 & 5). If these are less than 1/2, then by definition, the median will be the weighted median (line 6). If this is not the case, the weighted median on the side that is heaviar. The actual numbers on the lighter side do not change the situation. Only their sum of weights is important for determining the weighted median. So we consider the median to be having this weight in addition to its own weight and recursively try to find the weighted median over just the numbers in the heaviar side and the median together.

Lines $1, 6$ can be done in $O(1)$ time.
Lines $2, 3, 4, 5$ can be done in $O(n)$ time each.
Lines $6, 7$ take half the number of elements as input.

Therefore, the running time of this algorithm is $T(n) = T(n/2) + O(n) = O(n)$.

Another (similar) algorithm:

Linear-Weighted-Median(X,$start$,end,$sumleft$,$sumright$)

```
1. $i \leftarrow \lceil \frac{end - start}{2} \rceil$
2. $m \leftarrow SELECT(X, start, end, i)$
3. **for**$(j \leftarrow start$ to $i - 1)$
4.     $sumleft \leftarrow sumleft + X[j].weight$
5. **for**$(j \leftarrow i + 1$ to $end)$
6.     $sumright \leftarrow sumright + X[j].weight$
7. **if**$(sumleft \leq \frac{1}{2}$ AND $sumright \leq \frac{1}{2})$
8.     **return** $X[i]$
9. **if**$(sumleft > \frac{1}{2})$
10.    **return** Linear-Weighted-Median$(X, start, i, 0, sumright)$
11. **return** Linear-Weighted-Median$(X, i, end, sumleft, 0)$
```

This algorithm works much like Binary Search. First the median is checked to see if it is the weighted median; if so, it is returned. If not, this must be because one of the two inequalities defining the weighted median has been violated. If the first has been violated (ie, there is too much weight to the left of the median), then the weighted median must be further to the left side of the array, and so that left side is checked recursively for the weighted median. Similarly, if the second inequality has been violated, then the right side of the array is checked recursively for the weighted median.

To see that this algorithm runs in linear time, note that:
Line 1: $\Theta(1)$
Line 2: $O(n)$
Lines 3-6:$O(n)$
Lines 7-8:$\Theta(1)$
Lines 9-10: Recursion on an array of half the original size.

Thus, a recurrence relation describing the (worst-case) behavior of this algorithm is:
$T(n) = T(\frac{n}{2}) + O(n)$,
which can be shown to be $\Theta(n)$ by case 3 of the Master Theorem.