Chapter 9. Medians and Order Statistics

9.1-1.
Compare elements in tournament manner, that is, split them into pairs, compare two elements in each pair, and compare between the winners. We have to keep track of the result of each match. This is up to $n-1$ comparison in the worst case, deciding the smallest element. To find the second smallest element, find the smallest element from the $\lceil \frac{\lg n}{2} \rceil$ elements that lost to the smallest element. This consumes $\lceil \frac{\lg n}{2} \rceil - 1$ comparisons, resulting total $n + \lceil \frac{\lg n}{2} \rceil - 2$ comparisons in the worst case.

9.1-2.
Initially, there are n candidates for the maximum and minimum. At the end there should be just one candidate each, thus we must eliminate $2n-2$ candidates. Now classify the elements into one of the following four groups: max-min (if candidate for both), min (if candidate for only min), max (if candidate for only max), none (if neither). We construct the worst case using the following strategy:
1) max-min vs max-min : move the larger element to max, another one to min
2) max-min vs none : move the max-min element to max
3) min vs min : move the larger element to none
4) max vs max : move the smaller element to none
5) max vs max-min/min/none : do nothing
6) min vs max-min/max/none : do nothing
7) none vs none : do nothing
In this classification, case 1 can occur at most $\lfloor \frac{n}{2} \rfloor$ times because it removes 2 candidates from the max/min bin each time. Rest of cases can eliminate at most one candidate, and thus the number of comparisons is at least $(2n-2) - 2\lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{2} \rfloor = \lceil \frac{3n}{2} \rceil - 2$.

9.2-1.
Calling a zero-length array means either $p > q - 1$ or $q + 1 > r$. But if $p > q - 1$, then $q - p + 1 = k < 2$, but since $1 \leq i < k$, a contradiction. Similarly, if $q + 1 > r$, then $i > k = q - p + 1 > r - p$, which means $i \geq r - p + 2$, a condtradiction.

9.2-2.
$T(\max(k-1, n-k))$ is about chosing one partition from two. X_k is about chosing a pivot from the array, so those two procedures are independent.

9.2-3.

Algorithm 1: ITERATIVE-RANDOMIZED-SELECT(A, p, r, i)

```
while true do
    if p < r then
        return A[p];
    end
    q = RANDOMIZED-PARTITION(A, p, r);
    k = q - p + 1;
    if i == k then
        return A[q];
    else if i < k then
        r = q;
    else
        p = q;
        i = i - k;
    end
end
```

9.2-4.
When the partition always select the maximum element, we get the worst case. In this example, $[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$.

9.3-1.
Yes, it still works if they are divided into groups of 7, because we know the median of medians is greater than and less than roughly $\frac{2n}{7}$ of the elements. From this fact, we have

$$T(n) \leq T(\frac{n}{7}) + T(\frac{5n}{7}) + O(n).$$

Solving this gives $T(n) = O(n)$.
It does not work for groups of 3. In this case we have

$$T(n) \geq T(\frac{n}{3}) + T(\frac{2n}{3}) + O(n),$$

giving $T(n) = \Omega(n \lg n)$.

9.3-2.

$$n \geq 140 \Rightarrow \frac{3n}{10} - 6 \geq \frac{n}{4} + 1 \Rightarrow \frac{3n}{10} - 6 \geq \lceil \frac{n}{4} \rceil.$$

9.3-3.
If we modify quicksort to choose the pivot element to be the exact median every time, then this has the worst case running time $O(n \lg n)$ because selecting the pivot and partitioning the array consumes $O(n)$.

9.3-4.
Create a graph with n vertices and a draw a directed edge from vertex i to vertex j if the algorithm compared $A[i]$ and $A[j]$ and concluded that $A[i] < A[j]$. Let the ith order statistic of A to be x, then we can conclude that if there is a path from i to x, then $A[i] < x$. If there is a path to i from x, then $A[i] \geq x$. For every vertex i we have either a path from x or a path to x, because otherwise we cannot conclude that $A[i]$ is bigger than or smaller than x, so we cannot know that x is the i-th order statistic. Therefore, we can figure out the $i - 1$ smaller elements (which has a path to x) and $n - i$ larger elements (which has a path from x) without performing additional comparisons.

9.3-5.
Given the result of the index of the median, if the index is less than the half of the length of the array, then recurse on the first half. If the index is bigger, then recurse on the second half. If the index is precisely the half of the length, just return there.

9.3-6.

Algorithm 2: KTH-QUANTILES(A, p, r, i, j, k)

```
m = ⌊(i+j)/2⌋;
mid = ⌊mn/k⌋ − p + 1;
mid = SELECT(A, p, r, mid);
q = PARTITION(A, p, r, mid);
if i < m then
    L = KTH-QUANTILES(A, p, q - 1, i, m - 1, k);
end
if m < j then
    R = KTH-QUANTILES(A, q + 1, r, m + 1, j, k);
end
return L ∪ A[q] ∪ R;
```

9.3-7.
First, find the median in $O(n)$. Create a new array with the absolute difference with the median, and partition the array with the kth smallest number in $O(n)$. The smaller part is the desired set.

9.3-8.

Algorithm 3: MEDIAN-UNION(X, Y, n)

```
if n == 1 then
    return min(X[1], Y[1]);
end
k = ⌊n/2⌋;
if X[k] < Y[k] then
    return MEDIAN-UNION(X[k + 1, ··· , n], Y[1, ··· , k], k);
else
    return MEDIAN-UNION(X[1, ··· , k], Y[k + 1, ··· , n], k);
end
```

9.3-9.
We're basically minimizing $f(y) = \sum_{i=1}^{n} |y_i - y|$. Since $\frac{df}{dy} = \sum_{i=1}^{n} \operatorname{sgn}(y_i - y)$, this is zero only when the number of positive terms equals the number of negative terms, in other words, the number of wells above the spur pipeline equals the number of wells below the spur pipeline. Hence, if n is odd, the optimal solution is the median of all the y-coordinates of the wells. If n is even, the optimal solution is anything between the y coordinates of $\lceil \frac{n+1}{2} \rceil$th well and $\lfloor \frac{n+1}{2} \rfloor$th well. Both can be found in linear time.

9-1.
a. Sorting $O(n \lg n)$ + listing $O(i) = O(n \lg n)$.
b. Building a max-priority queue $O(n)$ + EXTRACT-MAX $O(\lg n)$ i times $= O(n + i \lg n)$.
c. Find the ith largest number and partition around that number $O(n)$ + sort the i largest number $O(i \lg i) = O(n + i \lg i)$.

9-2.
a. Let s_k the number of elements smaller than x_k. When all weights $w_i = \frac{1}{n}$, we have

$$\sum_{x_i<x_k} w_i = \frac{s_k}{n}, \sum_{x_i>x_k} w_i = \frac{n - s_k - 1}{2}.$$

The only value of s_k which satisfies the criteria is $s_k = \lceil \frac{n}{2} \rceil - 1$, making x_k to be the median.
b. Sort x_i in $O(n \lg n)$ time and compute the cumulative sum of weights S_i until you reach k such that $S_{k-1} < \frac{1}{2}$ and $S_k \geq \frac{1}{2}$. The weighted median is x_k.
c. Similar with SELECT, but we instead pass the desired partition weights (initially $\frac{1}{2}$). Find a median in linear time and partition around it. Sum the weights in the lower part and the upper part of the array. Check if the conditions fulfill. If they fulfill the conditions, we have the desired weighted median. If not, subtract $\frac{1}{2}$ from the bigger part and recurse with the smaller part and the subtracted number.
d. We're basically minimizing $f(p) = \sum_{i=1}^{n} w_i|p_i - p|$. Since $\frac{df}{dp} = \sum_{i=1}^{n} w_i \text{sgn}(p_i - p)$, this is zero only when $\sum_{x_i<x_k} w_i < \frac{1}{2}$ and $\sum_{x_i>x_k} w_i \leq \frac{1}{2}$. Hence the optimal solution is the weighted median.
e. Since the two dimensions can be treated independently each other, we can find the optimal solution by finding the weighted median x_k of the x-coordinates and the weighted median y_l of the y-coordinates respectively. (x_k, y_l) is the desired solution.

9-3.
a. When $i \geq \frac{n}{2}$, just call SELECT. Otherwise, let $m = \lfloor \frac{n}{2} \rfloor$, and divide the input with $A[p+1, \cdots, p+m]$, $A[p+m+1, \cdots, p+2m]$, and possibly $A[p+n]$. Compare $A[p+i]$ and $A[p+i+m]$, swap elements in each pair to place the bigger one into the left part and place the smaller one into the right part.

Now recursively find the ith smallest element in the right part, but this time, when swapping two elements in the right part, swap two elements in the left counterpart as well. This ensures that the i-th element of the whole array must be either in $A[p+1, \cdots, p+i]$ or $A[p+1+m, \cdots, p+i+m]$. Now collect the two subarrays $A[p+1, \cdots, p+i]$ and $A[p+1+m, \cdots, p+i+m]$ into a single array with $2i$ elements and use SELECT on that array.
b. For $i < \frac{n}{2}$, we shall show that for by substitution

$$U_i(n) \leq n + c_1 T(2i) \lg(\frac{n}{i}) - c_2 \lg(\lg n) T(2i),$$

that is, $U_i(n) = n + O(T(2i) \lg \frac{n}{i})$ for some positive constant c_1, c_2 and $n \geq 4$. We have

$$\begin{aligned} U_i(n) &= \lfloor \frac{n}{2} \rfloor + U_i(\lceil \frac{n}{2} \rceil) + T(2i) \\ &\leq \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil + c_1 T(2i) \lg \lceil \frac{n}{2} \rceil - c_1 T(2i) \lg i - c_2 \lg(\lg \lceil \frac{n}{2} \rceil) T(2i) \\ &\leq n + c_1 T(2i) \lg(\frac{n}{2} + 1) - c_1 T(2i) \lg i - c_2 \lg(\lg n - 1) T(2i) \\ &\leq n + c_1 T(2i) \lg n - c_1 T(2i) \lg i - c_2 \lg(\lg n) T(2i), \end{aligned}$$

provided that

$$c_1 T(2i) \lg(\frac{n}{2} + 1) - c_2 \lg(\lg n - 1) T(2i) \leq c_1 T(2i) \lg n - d(\lg \lg n) T(2i)$$

$$\Leftrightarrow c_1(\lg(\frac{n+2}{2n})) \leq c_2 \lg \frac{\lg n - 1}{\lg n}.$$

If we choose $c_2 \leq c_1 \lg \frac{4}{3}$, the inequality is satisfied.
c. When i is a constant, $T(2i) = O(1)$ and $\lg \frac{n}{i} = O(\lg n)$. Therefore, we have

$$U_i(n) = n + O(T(2i) \lg(\frac{n}{i})) = n + O(\lg n).$$

d. If $k = 2$,

$$U_i(n) = T(n) = n + T(n) - n \leq n + T(2i) - n = n + O(T(\frac{2n}{k}) \lg k).$$

If $k > 2$,

$$U_i(n) = n + O(T(2i) \lg(\frac{n}{i})) = n + O(T(\frac{2n}{k}) \lg k).$$

9-4.

a. z_i and z_j is compared when one of them is the first element picked as a pivot in the smallest interval containing i, j, k. The value of $\mathbb{E}[X_{ijk}]$ is $\frac{2}{k-i+1}$ if $i < j \leq k$, $\frac{2}{j-i+1}$ if $i \leq k \leq j$, $\frac{2}{j-k+1}$ if $k \leq i < j$.

b.

$$\mathbb{E}[X_k] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}[X_{ijk}] = \sum_{i=1}^{k}\sum_{j=i+1}^{n}\mathbb{E}[X_{ijk}] + \sum_{i=k+1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}[X_{ijk}]$$

$$= \sum_{i=1}^{k}\sum_{j=i+1}^{k-1}\mathbb{E}[X_{ijk}] + \sum_{i=1}^{k}\sum_{j=k}^{n}\mathbb{E}[X_{ijk}] + \sum_{i=k+1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}[X_{ijk}]$$

$$= \sum_{i=1}^{k-2}\sum_{j=i+1}^{k-1}\frac{2}{k-i+1} + \sum_{i=1}^{k}\sum_{j=k}^{n}\frac{2}{j-i+1} + \sum_{i=k+1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-k+1}$$

$$= \sum_{i=1}^{k-2}\frac{2(k-i-1)}{k-i+1} + \sum_{i=1}^{k}\sum_{j=k}^{n}\frac{2}{j-i+1} + \sum_{j=k+2}^{n}\sum_{i=k+1}^{j-1}\frac{2}{j-k+1}$$

$$= \sum_{i=1}^{k-2}\frac{2(k-i-1)}{k-i+1} + \sum_{i=1}^{k}\sum_{j=k}^{n}\frac{2}{j-i+1} + \sum_{j=k+2}^{n}\frac{2(j-k-1)}{j-k+1}$$

$$\leq 2(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}).$$

c.

$$\mathbb{E}[X_k] \leq 2(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1})$$

$$\leq 2(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{j=k+1}^{n}1 + \sum_{i=1}^{k-2}1)$$

$$= 2(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + (n-k) + (k-2)) \leq 2n + 2\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}$$

$$= 2n + 2\sum_{i=1}^{k}(\frac{1}{k-i+1} + \cdots + \frac{1}{n-i+1})$$

$$= 2n + 2[(\frac{1}{k} + \cdots + \frac{1}{n}) + (\frac{1}{k-1} + \cdots + \frac{1}{n-1}) + \cdots + (\frac{1}{1} + \cdots + \frac{1}{n-k+1})]$$

$$\leq 2n + 2[1 \cdot \frac{1}{1} + 2 \cdot \frac{1}{2} + \cdots + n \cdot \frac{1}{n}] = 2n + 2n = 4n.$$

d. The number of operations in RANDOMIZED-SELECT is linear to the number of comparisons, which is bounded by a linear function, so the expected running time is linear.