Chapter 5

Order Statistics

Exercise 5.1 *(Cormen et al. [4] Problem 9-2 p. 225)*

For n distinct elements $x_1, x_2, \cdots, x_n$ with positive weights $w_1, w_2, \cdots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the ***weighted (lower) median*** *is the element x_k satisfying*

$$\sum_{x_i < x_k} w_i < \frac{1}{2} \qquad \text{and} \qquad \sum_{x_i > x_k} w_i \leq \frac{1}{2}$$

For example, if the elements are $0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2$ and each element equal its weight (that is, $w_i = x_i$ for $i = 1, 2, \cdots, 7$), then the median is 0.1, but the weighted median is 0.2.

a. *Argue that the median of $x_1, x_2, \cdots, x_n$ is the weighted median of the x_i with weights $w_i = 1/n$ for $i = 1, 2, \cdots, n$.*

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

c. *Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm such as SELECT from Section 9.3 of Cormen et al. [3].*

d. *The post-office location problem is defined as follows. We are given n points $p_1, p_2, ..., p_n$ with associated weights $w_1, w_2, ..., w_n$. We wish to find a point p (not necessarily one of the input points) that minimizes the sum $\sum_{i=1}^{n} w_i d(p, p_i)$, where $d(a, b)$ is the distance between points a and b. Argue that the weighted median is a best solution for the one-dimensional post-office location problem, in which points are simply real numbers and the distance between points a and b is $d(a, b) = |a - b|$.*

e. *Find the best solution for the two-dimensional post-office location problem, in which the points are (x, y) coordinate pairs and the distance between points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is the Manhattan distance given by $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$.*

Solution

a. Let x_k be the median of $x_1, x_2, ..., x_n$. By the definition of median, x_k is larger than exactly

$\left\lfloor \frac{n+1}{2} \right\rfloor - 1$ other elements x_i. Then the sum of the weights of elements less than x_k is

$$\begin{aligned}\sum_{x_i<x_k} w_i &= \frac{1}{n}.\left(\left\lfloor \frac{n+1}{2} \right\rfloor - 1\right) \\ &= \frac{1}{n}.\left\lfloor \frac{n-1}{2} \right\rfloor \\ &\leq \frac{n-1}{2n} < \frac{n}{2n} < \frac{1}{2}.\end{aligned}$$

Since all the elements are distinct, x_k is also smaller than exactly $n - \left\lfloor \frac{n+1}{2} \right\rfloor$ other elements. Therefore

$$\begin{aligned}\sum_{x_i>x_k} w_i &= \frac{1}{n}.\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor\right) \\ &= 1 - \frac{1}{n}.\left\lfloor \frac{n+1}{2} \right\rfloor \\ &\leq 1 - \left(\frac{1}{n}\right)\left(\frac{n}{2}\right) \leq \frac{1}{2}.\end{aligned}$$

Therefore by the definition of weighted median, x_k is also the weighted median.

b. To compute the weighted median of n elements, we sort the elements and then sum up the weights of the elements until we have found the median.

Algorithm 4 WEIGHTED-MEDIAN(A)

```
1: k ← 1
2: s ← 0            ▷ s = total weight of all x_i < x_k
3: while s + w_k < 1/2 do
4:     s ← s + w_k
5:     k ← k + 1
6: end while
```

The loop invariant of this algorithm is that s is the sum of the weights of all elements less than x_k :

$$s = \sum_{x_i<x_k} w_i.$$

We prove this is true by induction. The base case is true because in the first iteration $s = 0$. Since the list is sorted, for all $i < k$, $x_i < x_k$. By induction, s is correct because in every iteration through the loop increases by the weight of the next element. The loop is guaranteed to terminate because the sum of the weights of all elements is 1. We prove that when the loop terminates x_k is the weighted median using the definition of weighted median. Let s' be the value of sat the start

of the next to last iteration of the loop: $s = s' + w_{k-1}$. Since the next to last iteration did not meet the termination condition, we know that

$$s' + w_{k-1} < \frac{1}{2} \qquad \text{and} \qquad \sum_{x_i < x_k} w_i = s < \frac{1}{2}$$

Note that if the loop has zero iterations this is still true since $s = 0 < 1/2$. This proves the first condition for being a weighted median. Next we prove the second condition. The sum of the weights of elements greater than x_k is

$$\sum_{x_i > x_k} w_i = 1 - \left(\sum_{x_i < x_k} w_i \right) - w_k = 1 - s - w_k$$

By the loop terminal condition

$$\sum_{x_i < x_k} w_i = s \geq \frac{1}{2} - w_k, \qquad \text{and then} - s \leq -\frac{1}{2} + w_k \quad \Rightarrow 1 - s \leq \frac{1}{2} + w_k.$$

It follows that: $1 - s - w_k \leq \frac{1}{2}$, and then

$$\sum_{x_i > x_k} w_k \leq \frac{1}{2}.$$

Thus x_k also satisfies the second condition for being the weighted median. Therefore x_k is the median and the algorithm is correct. The running time of the algorithm is the time required to sort the array plus the time required to find the median. We can sort the array in $O(n \log n)$ time. The loop in WEIGHTED-MEDIAN has $O(n)$ iterations requiring $O(1)$ time per iteration, so the overall running time is $O(n \log n)$.

c. The idea is to show that the weighted median can be computed in $\Theta(n)$ worst case time assuming we are given a $\Theta(n)$ time median algorithm. Assume that in case there is an even number of elements, that the median is the lower median. The basic strategy is similar to a binary search: the proposed algorithm, called LINEAR-TIME-WEIGHTED-MEDIAN(A), computes the **median** and recurses on the half of the input that contains the **weighted median**.

The weighted-median algorithm works as follows. If $n \leq 2$, we just return the brute-force solution. Otherwise, we proceed as follows. We find the actual median x_m of the n elements and then partition around it. We then compute the total weights of the two halves. If the weights of the two halves are each strictly less than $1/2$, then the weighted median is x_m. Otherwise, the weighted median should be in the half with total weight exceeding $1/2$. The total weight of the light half is lumped into the weight of x_m, and the search continues within the half that weighs more than $1/2$. Heres pseudocode, which takes as input a set $X = \{x_1, x_2, ..., x_n\}$.

Remember that we assume all elements to be distinct.

Algorithm 5 LINEAR-TIME-WEIGHTED-MEDIAN(X)

if length$[X] = 1$ **then**
  return x_1
else
  if length$[X] = 2$ **then**
    if $w_1 \geq w_2$ **then**
      Return x_1
    else
      Return x_2
    end if
  end if
end if
$n \leftarrow$ length$[X]$
$x_m \leftarrow$ MEDIAN(X)
Partition the set X around m
Compute $L = \{i : x_i < x_m\}$ and $R = \{i : x_i > x_m\}$
Compute $w_{\text{L}} = \sum_{i:x_i<x_m} w_i$ and $w_{\text{R}} = \sum_{i:x_i>x_m} w_i$
if $w_{\text{L}} < \frac{1}{2}$ and $w_{\text{R}} < \frac{1}{2}$ **then**
  Return m
else
  if $w_{\text{L}} > \frac{1}{2}$ **then**
    $w_m \leftarrow w_m + w_{\text{R}}$
    LINEAR-TIME-WEIGHTED-MEDIAN($L \cup \{x_m\}$)
  else
    $w_m \leftarrow w_m + w_{\text{L}}$
    LINEAR-TIME-WEIGHTED-MEDIAN($R \cup \{x_m\}$)
  end if
end if

The initial call is LINEAR-TIME-WEIGHTED-MEDIAN(X). Before the first recursive call happens, L contains all elements less than the median, R contains all elements greater or equal to the median, and w_L is the total weight (sum of the weights) of the elements in L.

Illustration of the LINEAR-TIME-WEIGHTED-MEDIAN algorithm on a first example.

x	5	7	3	9	2
w	0.15	0.1	0.2	0.3	0.25

First call to LINEAR-TIME-WEIGHTED-MEDIAN algorithm: LINEAR-TIME-WEIGHTED-MEDIAN(X)

$m \leftarrow$ median $= 5$
$L = \{2, 3\}$; $w_B = 0.45$
$R = \{7, 9\}$; $w_R = 0.4$
$w_L = 0.45 < 1/2$ and $w_R < 1/2$, therefore, weighted median $= x_1 = 5$.

Illustration of the LINEAR-TIME-WEIGHTED-MEDIAN algorithm on a second example.

x	5	7	3	9	2
w	0.25	0.2	0.05	0.4	0.1

First call to LINEAR-TIME-WEIGHTED-MEDIAN algorithm: LINEAR-TIME-WEIGHTED-MEDIAN(X)

$m \leftarrow$ median $= 5$
$L = \{2, 3\}$; $w_L = 0.15$
$R = \{7, 9\}$; $w_R = 0.6$
$w_L = 0.15 < 1/2$ and $w_R > 1/2$, therefore we iterate with a second call LINEAR-TIME-WEIGHTED-MEDIAN algorithm: LINEAR-TIME-WEIGHTED-MEDIAN($R \cup \{x_m\}$) after resetting the weight w_m to $w_m + w_L = .4$.

$m \leftarrow$ median $= x_2 = 7$
$L = \{5\}$; $w_L = 0.4$
$R = \{9\}$; $w_L = 0.4$
$w_L = 0.4 < 1/2$ and $w_R = 0.4 < 1/2$, therefore, weighted median $= x_2 = 7$.

To prove that the LINEAR-TIME-WEIGHTED-MEDIAN algorithm is correct, we show that the following precondition holds for every recursive call:
The weighted median x_m of the initial X is always present in the recursive calls of X, and w_m is the total weight of all elements x_i less than all the elements of X.

The precondition is trivially true for the initial call. We next prove that the precondition is also true in every recursive call by induction.

Assume for induction that the precondition is true. First let us consider the case in which $w_m + w_L > 1/2$. Let m be the median of X. Since x_m must be in X, x_m must be either in L or R. Since the total weight of all elements less than any element in R is greater than $1/2$, then by

definition the weighted median cannot be in R, so it must be in L. Furthermore, we have not discarded any elements less than any element in L, so l is correct and the precondition is satisfied. If $w_m + w_L \leq 1/2$, then x_m must be in R. All elements of R are greater than all elements of L, so the total weight of the elements less than the elements of R is $w_m + w_L$ and the precondition of the recursive call is also satisfied. Therefore by induction the precondition is always true. This algorithm always terminates because the size of X decreases for every recursive call. When the algorithm terminates, the result is correct. Since the weighted median is always in X, then when only one element remains it must be the weighted median. The algorithm runs in $\Theta(n)$ time.

Computing the median and splitting X into L and R takes $\Theta(n)$time. Each recursive call reduces the size of the array from n to $\lceil n/2 \rceil$. Therefore the recurrence is:

$$T(n) = T(n/2) + \Theta(n) = \Theta(n).$$

d. We argue that the solution to the one-dimensional post-office location problem is the weighted median of the points. The objective of the post-office location problem is to choose p in order to minimize the cost

$$c(p) = \sum_{i=1}^{n} w_i d(p, p_i) = \sum_{i=1}^{n} w_i |p - p_i|.$$

We can rewrite $c(p)$ as the sum of the cost contributed by points less than and points greater than p:

$$c(p) = \left(\sum_{p_i < p} w_i (p - p_i) \right) + \left(\sum_{p_i > p} w_i (p_i - p) \right).$$

Note that if $p = p_k$ for some k, then that point does not contribute to the cost. This cost function is continuous because $\lim_{p \to x} c(p) = c(x)$ for all x. To find the minima of this function, we take the derivative with respect to p:

$$\frac{dc}{dp} = \left(\sum_{p_i < p} w_i \right) - \left(\sum_{p_i > p} w_i \right).$$

Note that this derivative is undefined where $p = p_i$ for some i because the left-and right-hand limits of $c(p)$ differ. Note also that $\frac{dc}{dp}$ is a non-decreasing function because as p increases, the number of points $p_i < p$ cannot decrease. Note that $\frac{dc}{dp} < 0$ for $p < \min\{p_1, p_2, ..., p_n\}$ and $\frac{dc}{dp} > 0$ for $p > \max\{p_1, p_2, ..., p_n\}$. Therefore, there is some point p^* such that $\frac{dc}{dp} \leq 0$ for points $p < p^*$ and $\frac{dc}{dp} \geq 0$ for points $p > p^*$, and this point is a global minimum. We show that the weighted median y is such a point. For all points $p < y$ where p is not the weighted median and $p \neq p_i$ for some i,

$$\sum_{p_i < p} w_i < \sum_{p_i > p} w_i.$$

This implies that $\frac{dc}{dp} < 0$. Similarly, for points $p > y$ where p is not the weighted median and $p \neq p_i$ for some i,

$$\sum_{p_i<p} w_i > \sum_{p_i>p} w_i.$$

This implies that $\frac{dc}{dp} > 0$. For the cases where $p = p_i$ for some i and $p \neq y$, both the left-and right-hand limits of $\frac{dc}{dp}$always have the same sign so the same argument applies. Therefore $c(p) > c(y)$ for all p that are not the weighted median, so the weighted median y is a global minimum.

e. Solving the 2-dimensional post-office location problem using Manhattan distance is equivalent to solving the one-dimensional post-office location problem separately for each dimension. Let the solution be $p = (p_x, p_y)$. Notice that using Manhattan distance we can write the cost function as the sum of two one-dimensional post-office location cost functions as follows:

$$g(p) = \left(\sum_{i=1}^{n} w_i|x_i - p_x|\right) + \left(\sum_{i=1}^{n} w_i|y_i - p_y|\right)$$

Notice also that $\frac{\partial g}{\partial p_x}$ does not depend on the y coordinates of the input points and has exactly the same form as $\frac{dc}{dp}$ from the previous part using only the x coordinates as input. Similarly, $\frac{\partial g}{\partial p_x}$ depends only on the y coordinate. Therefore to minimize $g(p)$, we can minimize the cost for the two dimensions independently. The optimal solution to the two dimensional problem is to let p_x be the solution to the one-dimensional post-office location problem for inputs $x_1, x_2, ..., x_n$, and p_y be the solution to the one-dimensional post-office location problem for inputs $y_1, y_2, ..., y_n$.

Exercise 5.2 *(Cormen et al. [4] Exercise 9.3-6, p. 323)*

The k^{th} quantiles of an n-element set are the $k-1$ order statistics that divide the sorted set into k equal-sized sets (to within 1). Propose an $O(n \log k)$-time algorithm to list the k^{th} quantiles of a set.

Example 1. $A = \{2, 7, 4, 19, 6, 24, 1, 9, 34, 16, 12, 5\}$. $n = 12$. $k = 3$. *Three equal-sized sets:*

$$A_1 = \{2, 4, 1, 5\}; A_2 = \{7, 9, 6, 12\} : A_3 = \{34, 16, 24, 19\}$$

such that any element of A_1 (resp. A_2) is smaller or equal to any element of A_2 (resp.A_3).

If $k = k_1 = 3$, we obtain 3 sets, with four element each, and we are interested in the 5th order statistic (first item of the second set), and the 9th order statistic (first item of the third set).

If $k = k_2 = 2$, we obtain 2 sets, with six element each, and we are interested in the 7^{th} order statistic (first item of the second set).

The i^{th} order statistics of interest for computing the k^{th} quantiles are:

$$i_1 = \left\lfloor \frac{n}{k} \right\rfloor, \; i_2 = \left\lfloor \frac{2n}{k} \right\rfloor, \; \ldots, \; i_{k-1} = \left\lfloor \frac{(k-1)n}{k} \right\rfloor.$$

Alternatively, we could define them as

$$i_1 = \left\lceil \frac{n}{k} \right\rceil \; , \; i_2 = \left\lceil 2\frac{n}{k} \right\rceil \; , \; \ldots \; i_{k-1} = \left\lceil \frac{(k-1)n}{k} \right\rceil.$$

Assuming n is a power of k, a way to solve the problem is to use the RANDOMIZED-SELECT algorithm to select the position $i_{\lfloor \frac{k}{2} \rfloor}$. Because of nature of RANDOMIZED-SELECT, we know that $i_1 \ldots i_{\lfloor \frac{k}{2} \rfloor - 1}$ belongs to $1 \ldots (i_{\lfloor \frac{k}{2} \rfloor} - 1)$ and $i_{\lfloor \frac{k}{2} \rfloor + 1} \ldots i_{k-1}$ belongs to $(i_{\lfloor \frac{k}{2} \rfloor} + 1) \ldots n$. Then, we can recursively RANDOMIZED-SELECT on two intervals $1 \ldots (i_{\lfloor \frac{k}{2} \rfloor} - 1)$ and $(i_{\lfloor \frac{k}{2} \rfloor} + 1) \ldots n$.

Example. If $n = 256 = 4^4$ and we are interested in the 4^{th} quantiles, i.e., 64^{th}, 128^{th} and 192^{th} order statistics. We first compute 128^{th} order statistics, and only need to re-compute $\frac{n}{4}^{\text{th}}$ order statistics on the two remaining subsets (of size $\frac{n}{2}$ each) to get the two other order statistics we are looking for.

Assume now k is an arbitrary value. A way to solve this problem is to search first for the i^{th} order statistics such that $i = \ell \lfloor \frac{n}{k} \rfloor$ with ℓ such that i is the closed possible value to $\frac{n}{2}$ (either larger or smaller). i^{th} order statistics is then recursively computed on the two sublists of almost the same size, until all $\ell \lfloor \frac{n}{k} \rfloor^{\text{th}}$ order statistics, for all ℓ, are computed.

Example 2. If $n = 256$ and $k = 7$. The i^{th} order statistics of interest for computing the 7^{th} quantiles are:

$$i_1 = \lfloor \frac{256}{7} \rfloor = 36,\ i_2 = \lfloor 2\frac{256}{7} \rfloor = 73,\ i_3 = \lfloor 3\frac{256}{7} \rfloor = 109,$$
$$i_4 = \lfloor 4\frac{256}{7} \rfloor = 146,\ i_5 = \lfloor 5\frac{256}{7} \rfloor = 182,\ i_6 = \lfloor 6\frac{256}{7} \rfloor = 219.$$

The closest i to $n/2 = 128$ is i_3. We therefore first compute the 109^{th} order statistics, and then go on on each half or so, i.e., 36^{th} order statistics on a list of 108 elements (first half), and $(146 - 109)^{\text{th}} = 37^{\text{th}}$ order statistics on a list of $256 - 109 = 147$ elements.

In order to ease the presentation, we will assume we are in the first case, i.e., when k is an arbitrary value. We therefore need an algorithm that lists $i_u^{\text{th}}, \ldots, i_v^{\text{th}}$ quantiles of an array from position $A[p]$ to position $A[r]$.

Procedure LIST-QUANTILES(p, r, u, v)
$h = \lceil \frac{v - u + 1}{2} \rceil$
if $h = 0$ **then**
  return
end if
POS$= (u + h - 1)\lceil \frac{n}{k} \rceil$
print RANDOMIZED-SELECT$(\ A, p, r,$ POS $)$
LIST-QUANTILES$(p,$ POS $- 1, u, h - 1)$
LIST-QUANTILES$($POS $+ 1, r, h + 1, v)$
return
End Procedure

Back to Example 2, it means we first call LIST-QUANTILES$(1, 256, 1, 6, 7)$, as $n = 256$ and $k = 7$. We then get $h = 3$, that is we first compute the 3rd quantile, that is the set starting at the $i_3 =$ POS $= \lfloor 3\frac{256}{7} \rfloor = 109$th order statistics.
This leads to a first ith order statistics value, i.e., $i_3 = 109$, and we do not need to consider that value anymore.
We recurse on two subintervals, that exclude the POS value.
First recursion is: LIST-QUANTILES$(p,$ POS $- 1, u, h - 1)$ = LIST-QUANTILES$(1, 108, 1, 2)$. In other words, we recurse on the first sublist from element 1 to element 108, and we are interested in the first and second order statistics values.
Second recursion is: LIST-QUANTILES$($POS$+1, r, h+1, v)$ = LIST-QUANTILES$(110, 256, 4, 5)$. In other words, we recurse on the second sublist from element 1110 to element 256, and we are interested in the 4th and 5th order statistics values.

Let $T(n, k)$ be the number of operations of LIST-QUANTILES$(1, n, 1, k)$. From the above algorithm, we know that LIST-QUANTILES$(1, n, 1, k)$ calls RANDOMIZED-SELECT$(A, 1, n,$ POS$)$, LIST-QUANTILES$(1,$POS$-1, 1, \lfloor \frac{k}{2} \rfloor - 1)$ and LIST-QUANTILES$($POS $+ 1, n, \lfloor \frac{k}{2} \rfloor + 1, k)$.

Because RANDOMIZED-SELECT$(A, 1, n,$ POS$)= O(n)$ on average. Thus, we have following relation:

$$T(n,k) \leq T\left(\left\lfloor\frac{k}{2}\right\rfloor \times \left\lfloor\frac{n}{k}\right\rfloor - 1, \lfloor\frac{k}{2}\right\rfloor - 1\Big) + T\left(n - \left\lfloor\frac{k}{2}\right\rfloor \times \left\lfloor\frac{n}{k}\right\rfloor, k - \left\lfloor\frac{k}{2}\right\rfloor\right) + O(n).$$

Using the following approximation,

$$T\left(\left\lfloor\frac{k}{2}\right\rfloor \times \left\lfloor\frac{n}{k}\right\rfloor - 1, \left\lfloor\frac{k}{2}\right\rfloor - 1\right) \approx T\left(n - \left\lfloor\frac{k}{2}\right\rfloor \times \left\lfloor\frac{n}{k}\right\rfloor, k - \left\lfloor\frac{k}{2}\right\rfloor\right) \approx T\left(\frac{n}{2}, \frac{k}{2}\right),$$

we deduce the following simplified relation:

$$\begin{aligned} T(n,k) \quad & \leq 2T(\frac{n}{2}, \frac{k}{2}) + O(n) \\ & \leq 2^2 T(\frac{n}{2^2}, \frac{k}{2^2}) + O(n) + O(\frac{n}{2}) \\ & \leq \dots \\ & \leq kT(\frac{n}{k}, 1) + O(n) + O(\frac{n}{2}) + O(\frac{n}{2^2}) + \ldots + O(\frac{n}{k/2}). \end{aligned}$$

We know that there are $\log k$ O-terms, moreover

$$O(n) = O\left(\frac{n}{2}\right) = O\left(\frac{n}{2^2}\right) = \ldots = O\left(\frac{n}{k/2}\right).$$

We thus deduce:

$$T(n,k) \leq kT\left(\frac{n}{k}, 1\right) + O(n \log k).$$

As $kT(\frac{n}{k}, 1) = k$, we then get: $T(n,k) = O(n \log k)$.

Exercise 5.3 *(Cormen et al. [4] Problem 9.3-9 p. 223)*

Professor Olay is consulting for an oil company, which is planning a large pipeline running east to west through an oil field of n wells. From each well, a spur pipeline is to be connected directly to the main pipeline along a shortest path (either north or south), as shown in Figure 5.1. Given $x-$ and $y-$ coordinates of the wells, how should the professor pick the optimal location of the main pipeline (the one that minimizes the total length of the spurs)? Show that the optimal location can be determined in linear time.

Figure 5.1: Professor Olay needs to determine the position of the east-west oil pipeline that minimizes the total length of the north-south spurs.

Solution

A first observation is that connection of the wells to the main pipeline depends **only** on the y coordinate in a coordinate system where the abscissa axis is parallel to the main pipeline. The overall length of the connections is then

$$\ell_{\text{connection}}(y) = \sum_{i=1}^{n} |y - y_i|.$$

Next, let us show that in order to find the optimal placement for Professor Olays pipeline, we need only find the median(s) of the y-coordinates of his oil wells. Indeed, we will now prove that: The optimal y-coordinate for Professor Olays east-west oil pipeline is as follows:

1. If n is even, then on either the oil well whose y-coordinate is the lower median or the one whose y-coordinate is the upper median, or anywhere between them.

2. If n is odd, then the oil well whose y-coordinate is the median.

Proof.
We examine two cases: n is even and then n is odd. In each case, we will start out with the pipeline at a particular y-coordinate and see what happens when we move it. We denote by s the sum of the north-south spurs with the pipeline at the starting location and s' will denote the sum after moving the pipeline.

Case 1. n is even. Let us start with the pipeline somewhere on or between the two oil wells whose y-coordinates are the lower and upper medians. If we move the pipeline by a vertical distance d without crossing either of the median wells, then $n/2$ of the wells become d farther from the pipeline and $n/2$ become d closer, and so $s' = s + d \times n/2 - d \times n/2 = s$; thus, all locations on or between the two medians are equally good.

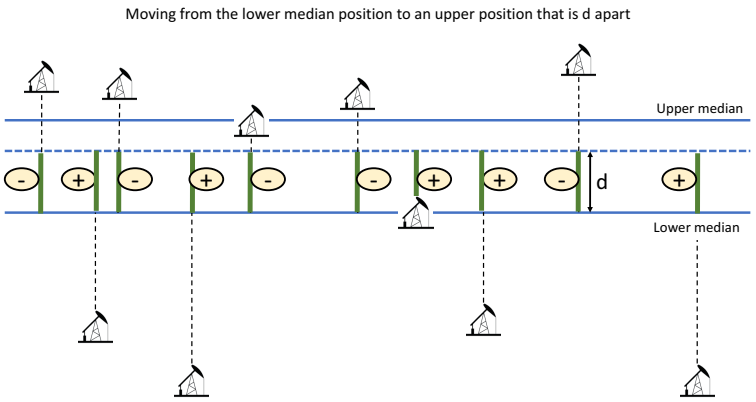

Figure 5.2: Case 1. n is even and the pipeline is positioned between the upper and the lower median.

Now suppose that the pipeline goes through the oil well whose y-coordinate is the upper median. What happens when we increase the y-coordinate of the pipeline by $d > 0$ units, so that it moves above the oil well that achieves the upper median? See Figure 5.2 for an illustration. All oil wells whose y-coordinates are at or below the upper median become d units farther from the pipeline, and there are at least $n/2+1$ such oil wells (the upper median, and every well at or below the lower median). There are at most $n/2 - 1$ oil wells whose y-coordinates are above the upper median, and each of these oil wells becomes at most d units closer to the pipeline when it moves up. Thus, we have a lower bound on $s' \geq s + d(n/2 + 1) - d(n/2 - 1) = s + 2d > s$.

We conclude that moving the pipeline up from the oil well at the upper median increases the total spur length. A symmetric argument shows that if we start with the pipeline going through the oil well whose y-coordinate is the lower median and move it down, then the total spur length increases. We see, therefore, that when n is even, an optimal placement of the pipeline is anywhere on or between the two medians.

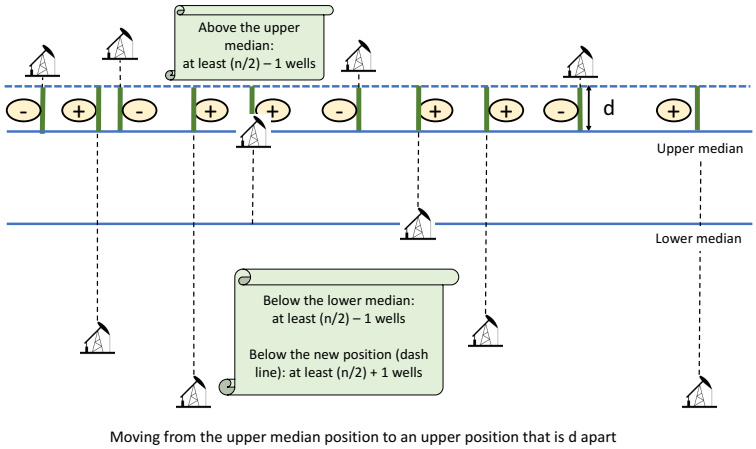

Figure 5.3: Case 1. n is even and the pipeline is positioned higher than the upper median.

Case 2. n is odd. We start with the pipeline going through the oil well whose y-coordinate is the median, and we consider what happens when we move it up by $d > 0$ units. All oil wells at or below the median become d units farther from the pipeline, and there are at least $(n+1)/2$ such wells (the one at the median and the $(n-1)/2$ at or below the median. There are at most (n - 1)/2 oil wells above the median, and each of these be- comes at most d units closer to the pipeline. We get a lower bound on s' of $s' \geq s + d(n+1)/2 = d(n-1)/2 = s + d > s$, and we conclude that moving the pipeline up from the oil well at the median increases the total spur length. A symmetric argument shows that moving the pipeline down from the median also increases the total spur length, and so the optimal placement of the pipeline is on the median.

Since we know we are looking for the median, we can use the linear-time median finding algorithm.

Complexity: $O(n)$ as computing the median element of a set of n elements takes $O(n)$ time.

Exercise 5.4 *(Manber [13] Exercise 6.38, p. 179)*

The goal is to find the kth element while minimizing the running time as well as using very little space (although not necessarily minimal space). The input is a sequence of elements $x_1, x_2, \ldots, x_n$, given one at a time. Design an $O(n)$ expected time algorithm to compute the kth element using only $O(k)$ memory cells. The value of k is known ahead of time (so that sufficient amount of memory can be allocated), but the value of n is not known until the last element is seen.

Solution

The idea is to consider $2k$ elements at a time. We start with the first $2k$ elements, and find their median. All elements greater than the median can be eliminated. We now look at the next k elements and do the same. This process requires approximately n/k phases, each consisting of computing the median of $2k$ elements.

Exercise 5.5 *It is said that the following problem has been used by Google as one of their interview questions. Let $S1$ and $S2$ be two arrays, each of size n, that are already sorted. Let's assume that all elements in the two arrays are distinct and n is a power of 2.*

(i) Design an $O(\log n)$-time algorithm to find the median in the union of the two arrays. Since there are $2 \times n$ elements in total, we define the median to be the n-th smallest element. [Hint: Compare the median of $S1$ and the median of $S2$. Then, what can you say about the median of $S1 \cup S2$ You can still get partial credits for this problem if you give a slower algorithm.

(ii) Design an $O(\log k)$-time algorithm to find the k-th smallest element in the union of the two arrays. You can use the solution for (a) even if you can't solve (i).

Solution

(i) If S1$[n/2]$ > S2$[n/2]$, there must be at least $n/2 - 1 + n/2 = n - 1$ elements smaller than S1$[n/2]$. This means that the n-th element in S1$\cup$ S2 must be in S1$[1..n/2]$ or S2$[n/2 + 1..n]$. In particular, it is exactly the $n/2$-th element in S1$[1..n/2]\cup$ S2$[n/2 + 1..n]$.

Note that this is exactly the same problem as before, except that the array size is now $n/2$. So we can recursively solve the problem.

Similarly, if S1$[n/2]$ < S2$[n/2]$, there must be at least $n - 1$ elements smaller than S2$[n/2]$. This means that the n-th element in S1 $\cup$ S2 must be in S1$[n/2 + 1..n]$ or S2$[1..n/2]$. Then, we recursively solve the problem on S1$[n/2 + 1..n]$ and S2$[1..n/2]$. We reach the base case when the array size is 1. Since we reduce the problem size by half each time, the running time is $O(\log n)$

(i) Notice that the k-th element must be from S1$[1..k]$ and S2$[1..k]$. Then we apply the solution in *(i)* on these two arrays. The running time is thus $O(\log k)$.