9.1.1 *Show that the second smalest of n elements can be found in $n + lgn$ $- 2$ comparisons in the worst case.* (***Hint Also find the smallest element***):

We can perform the following: Let form pairs based on the numbers, so we do $\frac{n}{2}$ comparisons between pairs we them end up with $\frac{n}{2}$ we again perform such task until we have the smallest Since each comparison result in an element being rejected as the smallest in this situation we perform n
$- 1$ comparisons. Now our smallest element was compared with at most the height of the three which is $\lfloor \log n \rfloor$ since we will not compare with the smallest anymore we reduce 1 comparison resulting in $n + \lceil logn \rceil - 2$

9.1.2 *Given $n > 2$ distinct numbers, you want to find a number that is neither the minimun nor the maximun. What is the smallest number of comparisons that you need to perform*?
The minimum number of comparisons for distinct numbers is 3. With one comparison You will only know if a number a is lower than a number b, but they can be the maximun and minimun. With two comparisons with 3 numbers you can specify a order betwen between 2 of them with 1 of them. in this case you can perform $a < b$ and $a > c$. In this case a is not the max *nor the min. But if $a > b$ and $a > c$ we can't conclude anything. three comparisons guarantee that we perform ar order in the three elements since they are distinct the middle one cannot be the* max *nor the min*

9.1.3 *A racetrack can run races with five horses at a time to determine their relative speeds. For 25 horses it takes 6 races to determine the fastest horse assuming transitivity. What is the minimum number of races it takes to determine the fastest three horses out of 25*?

We have 25 horses and we will divide them into groups of 5, lets call this groups $\boldsymbol{A_1, A_2, A_3, A_4, A_5}$. *We then perform a run in each of them. Totalizing 5 runs*
We then perform another run with the winners. Suppose, without loss of generality That A_1 contains the winner, A_2 the second and so on.. A_1. Let Denote A_{11} the winner of A_1 A_{21} the winner of A_2 and so on. We know that A_{41} and A_{51} cannot be in the three bests And so all theirs groups cannot. We already know the winner so we won't use it again in a race. A_{22} can be the third, but A_{23}, A_{24} and A_{25} cannot, since they are already worse 2 positions below their group leader. Based on this the possibly 2 fastest, with the first, are $A_{12}, A_{13}, A_{21}, A_{22}, A_{31}$ We just need one more run. so 7 runs is enough

---

9.1.4 *Prove the lower bound of $\left\lceil \frac{3n}{2} \right\rceil - 2$ comparisons in the worst case to find both the maximun and minimum of n numbers. Hint* (***Consider how many numbers are potentially either the maximum or minimum, and investigate how a comparison affects these coutns***)

Iniially any of the n elements can be MINIMUN or MAXIMUM. Lets create two sets MAX and MIN in which, initially we set MAX to be equal our n members and so does MIN. We then have 2 values , lets say c in MIN and MAX. We them compare a with b, we will get $a \leq b$ *or* $b \leq a$. *In either case we can remove two elements from the sets if* $a \leq b$ *we can take out a from MAX and b from MIN.*
Based on that if n is even, we can make pairs of comparisons with the different elemnts and in the end, after $\frac{n}{2}$ *comparisons our MAX set and MIN set will be disjoint with* $MIN\ U\ MAX = A$. *In this case we need to eliminate the elements from MAX comparing them since each comparisos allow us to remove just one element at a time we need to perform* $\frac{n}{2} - 1$ *comparisons, the same for the MIN.*
With that out total comparisons is $n - 2 + \frac{n}{2} = \frac{3n}{2} - 2$
Now if n is odd we cannot make pair wise comparisons exactly, We can reduce.
after $\frac{n-1}{2}$ *comparisons the size of each set to* $\frac{n+1}{2}$ *with one element shared between them now if we compare this element, say it c, with other of MIN, say it* min *and* $c \leq min$ *we can remove* min *from Min and c from MAX, the same goes for MAX*
Now if we compare and $min \leq c$ *we can only take out* min *from MIN*
lets'not compare this element by now. We them compare the $\frac{n-1}{2}$ *elements in MIN with that we can remove* $\frac{n-3}{2}$ *elements from MIN through* $\frac{n-3}{2}$ *comparisons, the same goes to MAX we them have realized* $\frac{n-1+n-3+n-3}{2} = \frac{3n-7}{2}$ *in this situation we have say, a and c in the MIN set and b and c in the MAX Set. There is two cases if we compare b and c and we get* $c \leq a$ *we take out c from MAX and a from MIN.*
So with one more comparisom we are done, leading us to $\frac{3n-5}{2}$ *comparisons*
if $a \leq c$ *we need to compare b with a and we are done.*
so $\frac{3n-3}{2}$ *comparisons are needed* $\rightarrow \frac{3n-1}{2} - 2$ *and* $\frac{3n+1}{2} - 2$ *comparisons needed*
So if n is odd the worst case we need $\frac{3n+1}{2} - 2$ *We know that*
$\frac{3n+1}{2}$ *is an integer and* $\left\lceil \frac{\mathbf{3n}}{\mathbf{2}} \right\rceil$ *too with* $\frac{3n}{2} \leq \left\lceil \frac{\mathbf{3n}}{\mathbf{2}} \right\rceil \leq \frac{3n+2}{2}$ *since they are integers*
And there can be only 1 integer betweem two non integers whose subtraction is one
we conclude that in the worst case our lower bound is $\left\lceil \frac{\mathbf{3n}}{\mathbf{2}} \right\rceil - \mathbf{2}$

9.2.1 *Show that RANDOMIZED − SELECT never makes a recursive call to a* 0 *−length array*.

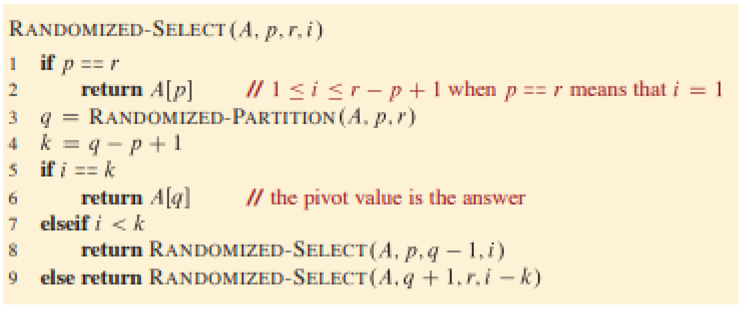
An array has at least one element if $p == r$, *to have a 0 length we need that* $p > r$
In this situation, we know that q is equal to p or at most r. So k is such that:
$1 \leq k \leq r - p + 1 = size\ of\ the\ array$
Now $q - 1$ *must be smaller than p, if q is* $p + 1$ *they are equal. if* $q = p$ *it's greater*
Now r must be smaller than $q + 1$, *thin only occur if* $q = r$

So we have this two situations when a possible length 0 *array would be called by RANDOMIZED − SELECT. Now if* $q = p$, *it means that* $k = 1$. *and will will call the length* 0 *array only if* $i < 1$. *What is not the case since* $1 \leq i \leq$ *size of the array.*
In the other case we would only call
the length 0 *array if* $q = r$ *so* $k = r - p + 1 =$ *size of the array and* i *would need to be higher than the size of the array, but one more time* $1 \leq i \leq$ *size of the array*

9.2.2 ***Write an iterative version of RANDOMIZED − SELECT***

```
RANDOMIZED − SELECT(A, p, r, i):
answer;
while true:
  if p == r
   return A[p]
  q = RANDOMIZED − PARTITION(A, p, r)
  k = q − p + 1
 if i == k
   answer = A[q]
   break
 else if i < k
   r = q − 1
 else:
   p = q + 1
   i = i − k
```

9.2.3 ***Suppose that RANDOMIZED − SELECT is used to select the minimum element of the array*** $A = \{2, 3, 0, 5, 7, 9, 1, 8, 6, 4\}$. ***Describe a sequence of partition that results in a worstcase perfromance of RANDOMIZED − SELECT.***



```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2      return A[p]    // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6      return A[q]    // the pivot value is the answer
7  elseif i < k
8      return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

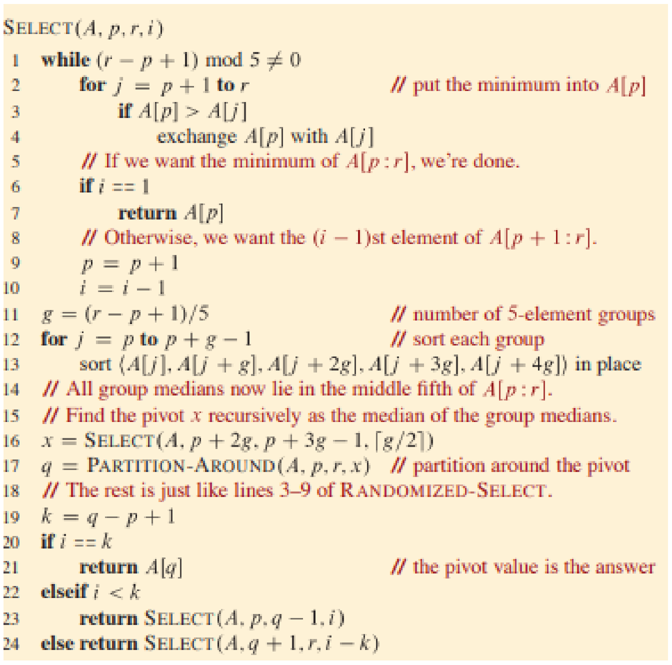
we want to select 0 *in the worst case possible so we realize a botom up approach, in this situation we want* 0 *to be the last element in the array, so we want to return it if* $p == r$
So the previous call must be for $i = 1$, *since* $i = 1$, i *cannot be higher than* k, *whose minimun value is* 1, *since* $i = 1$ *we cannot have* $k = 1$ *otherwise we would return in line* 6 *so* $i < k$, *what means that* $q - 1 = p \rightarrow q = p + 1$ *and* $k = 2$
that is the position of the pivot. The worst case we isolate 0 *as far as possible.*
We can select 9,8,7,6,5,4,3,2,1

9.2.4 *Argue that the expected running time of RANDOMIZED − SELECT does not depend on the order of the elemetns in its input array $A[p{:}r]$. That is, the expected runing time is the same for any permutation of the input array $A[p{:}r]$. (Hint: argue by indution on the length n of the input array)*

Let An array of size $= 1$ *obvisouly it will be independent. Now suppose it works for* $n - 1$*, lets prove it works for* n *The partitioning is independent, since its Uniformly random. After the selection of the pivot we have the array subdivide into 3 groups, basically and the running time of the pivot finding will be* $\Theta(n)$*. These groups are Independent of the order of the inptbut, simply in the intrinsic characteristics of the input Again the recursive callsare independent of ther order, but are dependent ofthe i −th smallest we want to find and the intrisic values of the input*

9.3.1

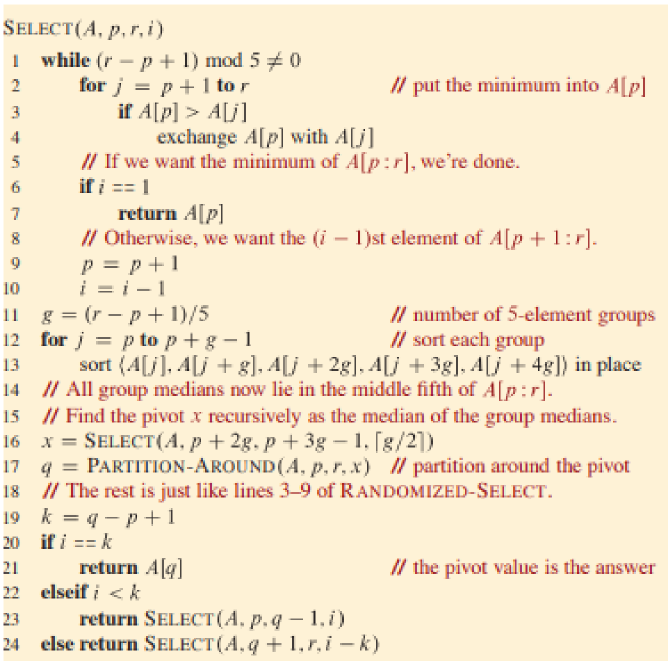
In the algorithm SELECT, the input elements are divided into groups of 5. Show that the algorithm works in linear time if the input elements are divided intogroupsof 7 instead of 5.



```
SELECT(A, p, r, i)
while (r − p + 1) mod 5 ≠ 0
    for j = p + 1 to r                        // put the minimum into A[p]
        if A[p] > A[j]
            exchange A[p] with A[j]
    // If we want the minimum of A[p : r], we're done.
    if i == 1
        return A[p]
    // Otherwise, we want the (i − 1)st element of A[p + 1 : r].
    p = p + 1
    i = i − 1
g = (r − p + 1)/5                             // number of 5-element groups
for j = p to p + g − 1                        // sort each group
    sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
// All group medians now lie in the middle fifth of A[p : r].
// Find the pivot x recursively as the median of the group medians.
x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)
q = PARTITION-AROUND(A, p, r, x)   // partition around the pivot
// The rest is just like lines 3–9 of RANDOMIZED-SELECT.
k = q − p + 1
if i == k
    return A[q]                               // the pivot value is the answer
elseif i < k
    return SELECT(A, p, q − 1, i)
else return SELECT(A, q + 1, r, i − k)
```

We basically need to adjust the proof provided in the book, by constants of 7. The number g of groups will be at most $\frac{n}{7}$*. There atleast* $4\left(\left\lceil\frac{g}{2}\right\rceil + 1\right) \geq 2g$ *elements greater than or equal to the pivot and* $4\left(\left\lceil\frac{g}{2}\right\rceil + 1\right) \geq 2g$ *elements less or equal to the pivot. what leave us with* $5g \leq \frac{5n}{7}$ *elements in the recursive call. The recurrences becomes* $T(n) \leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + O(n)$*. Using substitution method or akkra − Bazzi we can verify that is still linear*

9.3.2 *Suppose that the preprocessing in lines 1 − 10 of SELECT is replaced by a base case for $n \geq n_0$, where n_0 is a suitable contants;*
that g is chosen as $\left\lfloor \frac{r-p+1}{5} \right\rfloor$: and that the elements in $A[5g{:}n]$ belong to no group
Show that although the recurrence for the running time becomes messier it still solves to $\Theta(n)$



```
SELECT(A, p, r, i)
1  while (r − p + 1) mod 5 ≠ 0
2      for j = p + 1 to r                  // put the minimum into A[p]
3          if A[p] > A[j]
4              exchange A[p] with A[j]
5      // If we want the minimum of A[p : r], we're done.
6      if i == 1
7          return A[p]
8      // Otherwise, we want the (i − 1)st element of A[p + 1 : r].
9      p = p + 1
10     i = i − 1
11 g = (r − p + 1)/5                       // number of 5-element groups
12 for j = p to p + g − 1                  // sort each group
13     sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
14 // All group medians now lie in the middle fifth of A[p : r].
15 // Find the pivot x recursively as the median of the group medians.
16 x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)
17 q = PARTITION-AROUND(A, p, r, x)   // partition around the pivot
18 // The rest is just like lines 3–9 of RANDOMIZED-SELECT.
19 k = q − p + 1
20 if i == k
21     return A[q]                         // the pivot value is the answer
22 elseif i < k
23     return SELECT(A, p, q − 1, i)
24 else return SELECT(A, q + 1, r, i − k)
```

9.3.3 *Show how to use SELECT as a subroutine to make quicksort run in $O(n\lg n)$ time in the worst case, assuming that all elements are distinct*
Select returns the i − th smallest element of an array in linear time in the size of the array. In this situation we can guarantee a great partitioning
Based on that we can substitute the casual partitioning from
$PARTITION - AROUND\left(A, p, r, SELECT\left(p, r, \left\lfloor \frac{r-p+1}{2} \right\rfloor\right)\right)$
In this case we will always recurse on $T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n)$ *what is the* $O(n\lg n)$

9.3.4 *Suppose that an algorithm uses only comparisons to find the i −th smallest element in a set of n elements. Show that it can also find the i − 1 smaller elements and the n − i larger elements without performing any aditional comparison*

Lets say the sorted array is $A_1, A_2 \dots A_n$. The i smallets element is A_i, the $i - 1$ smaller is A_{i-1} and the $n - i$ larger. we know that A_n is the larger, A_{n-1} the second larger. A_{n-2} the third larger.. $A_{n-(n-i-1)}$ the $n - i$ larger $= A_{i+1}$.
So basically what this wants we also find the immediately smaller and immediately greater of the median. In the last operation we discover one them. Given all possible Comparisons we need to know that A_i is exactly higher than i elements Since we only do comparisons and nothing else, we can only have such information with we compare A_{i-1} and this be higher and compare with A_{i+1} and this be lower. So we need our median

to be compared with both. The last operation will be with one of them
Suppose that in the last operation we discovered the $i - 1$ *smallest. It can only be the lowest of the two, now, how do we have choosen the one to be compared with the* A_{i-1} *smaller? IN other words how we have choosen the median to be compared?*
We know it was compared with the A_{n+i} *larger in some moment. Now in our algorithm The* A_i *were compared to some numbers, let S be the set of elements compared to them That were greater, This set is not empty containing at least* A_{n+1} *and possibly other. The question is How can we know which of these elements is the* A_{n+1}*?*
After we find the median. There is only one way we need to compare with some who is lower than i elements. There is ony such element in the comparisons tree so we are done. There is no other way since we would need to keep

---

9.3.5 *Show how to determine the median of* 5 − *elements set using only* 6 *comparisons*

Lets say we have a, b, c, d, e. *we compare a with b and c with d and the largest of them Without loss of generality we end with* $a < b < d$ *and* $c < d$. *d cannot be median*
Now we compare c with e resulting in $c < e$ *or* $e < c$. *we now compare b with e or c resulting in* $a < b < e$ *and* $c < e$ | *or* $c < e < b$ *and* $a < b$ | *ora* $< b < c$ *and* $e < c$ | *or* $e < c < b$ *and* $a < b$. *We realized* 5 *comparisons until now.*
In the four cases above we endup with:
1. *e cannot be the median what leads us to* $a < b$ *and c*
2. *b cannot be the median what leads us to* $c < e$ *and a*
3. *c cannot be the median what leads us to* $a < b$ *and e*
4. *b canot be the median what leads us to* $e < c$ *and a*

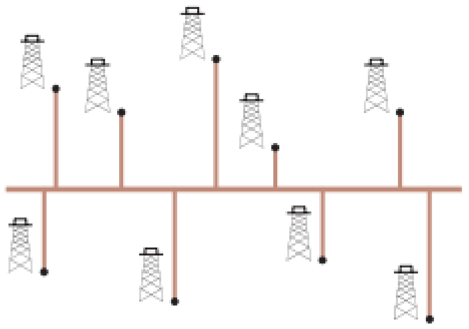
We eliminated two possible median that have 3 *or more less elements than them So we eliminated the greatest. Now our median must be someone in those three how is the largest. With that said there is only two possibilities i each case since we know that one of them is smaller than other*

9.3.6 You have black-box *worst case linear − time median subroutine. Give a simple , linear time algorithm tha solves the selection problem for an arbitrary order statistics*

We have an algorithm that always find the $\left\lfloor \frac{n+1}{2} \right\rfloor$ − *th order statistics of an array*
We want him to find any i order. We showed above that we also discover
$\left\lfloor \frac{n+1}{2} \right\rfloor + 1$ *and* $\left\lfloor \frac{n+1}{2} \right\rfloor - 1$ *we can remove this elements and recurse.*
Now the idea is as following, let $j = \left\lfloor \frac{n+1}{2} \right\rfloor - i$. *if* $j > 0$ *than*
if $-1 \le j \le 1$ *we already got the* i − *order statistc.*
if $j < -1$ *we want elements greater than* $\left\lfloor \frac{n+1}{2} \right\rfloor + 1$. *Since we are removing* 3 *elements lower we reduce i by* 3 *and recurse. If* $j > 1$ *we only remove the elements and recurse.*

$Now\ in\ each\ call\ we\ have\ a\ chance\ of\ \frac{3}{n}..\frac{3}{n-3}...\frac{3}{n-6}..\frac{3}{n-(n-3)}\ of\ achieving$
$the\ desirable\ index. The\ total\ number\ of\ calls\ are\ i \rightarrow$
$n-3(i-1)=3 \rightarrow\ \ n-3i+3=3 \rightarrow n=3i \rightarrow i=\frac{n}{3}\ so\ we\ have\ \frac{n}{3}\ worst\ case\ calls$
$And\ since\ we\ use\ linear\ time\ in\ each\ one\ of\ the, we\ have\ n+n-3+n-6\ ...$
$$\sum_{i=1}^{\frac{n}{3}} n-3\sum_{i=1}^{\frac{n}{3}} i=\frac{n^2}{3}-3\left(1+\frac{n}{2}\right)\frac{n}{2\cdot 2} \rightarrow its\ still\ O(n^2).$$
$Another\ approach\ is\ since\ you\ find\ the\ median\ in\ O(n) you\ can\ realize\ a\ partition$
$with\ cost\ O(n)\ and\ recurse\ on\ the\ desired\ region. Since\ its\ the\ median$
$we\ Reduce\ it\ to\ a\ recursions\ of\ T(n)=T\left(\frac{n}{2}\right)+O(n)=O(n)$

9.3.7Professor Olay is consulting for an oil company, which is planning a large pipeline running east to west through an oil ûeld of n wells. The company wants to connect a spur pipeline from each well directly to the main pipeline along a shortest route (either north or south), as shown in Figure 9.4. Given the x – and y – coordinates of the wells, how should the professor pick an optimal location of the main pipeline to minimize the total length of the spurs? Show how to determine an optimal location in linear time



$Let\ y_m\ be\ the\ location\ of\ the\ main\ pipeline.$
$Say\ it\ there\ are\ k\ towers\ above\ and\ and\ n-k\ below.$
$The\ total\ ammount\ is\ equal\ to\ the\ y\ distance\ from\ the\ main\ pipeline. In\ this\ case\ we\ have$
$$\sum_{i=1}^{k} Y_{ki}-Y_m+\sum_{i=1}^{n-k} -Y_{ni}+Y_m \rightarrow Y_m(n-k)-k(Y_m)+\sum_{i=1}^{k} Y_{ki}-\sum_{i=1}^{n-k} Y_{ni} \rightarrow$$
$Despite\ this\ analyzes\ we\ can\ use\ modules\ so\ the\ size(Y_m)=\sum_{i=1}^{n}\sqrt{(Y_i-Y_m)^2} \rightarrow$
$everything\ here\ is\ constant\ except\ Y_m, Lets\ find\ the\ minimun\ through\ the$
$derivative\ . If\ Y_m\ is\ greater\ than\ k\ and\ lower\ than\ k\ its\ derivative=0$
$Now\ lets\ Y_1\ be\ the\ lowest. for\ \ for\ Y_m<Y_k \rightarrow The\ derivative\ is\ lower\ than\ 0\ so\ the$
$derivative\ is\ negative. if\ Y_k \leq Y_m \leq Y_{k+1}\ the\ derivative\ is\ 0\ and\ \ if\ Y_m>Y_{k+1}$
$the\ derivative\ is >0\ So\ when\ we\ have\ a\ even\ number\ if\ Y_{\frac{n}{2}} \leq Y_m \leq Y_{\frac{n}{2}+1}$
$we\ have\ the\ minimu. Now\ if\ n\ is\ even\ let\ Y_m<Y_{\frac{n+1}{2}}\ the\ derivative\ is\ negative. if$
$Y_m=\ Y_{\frac{n+1}{2}}\ the\ derivative\ is\ 0\ and\ after\ that\ the\ derivative\ is\ greater\ than\ 0; Based$
$on\ that\ the\ best\ location\ to\ construct\ is\ the\ median. As\ we\ saw\ before\ we\ can\ use$
$SELECT\ algorithm\ to\ choose\ the\ median\ in\ n\ worst\ case$

9.3.8 *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). Given an O(nlog(k))time algorithm to list the k − th quantiles of a set.*

if k is even we can verif that the $\frac{k}{2} \cdot \frac{n}{k}$ *is the median and also its the* $\frac{k}{2} -$ *th quantile*
In the case k is odd we catch the $\frac{k-1}{2}$ *quantile and then divide the arrays and recurse*
So to the following use the SELECT subroutine to find the $\left\lfloor \frac{k}{2} \right\rfloor$ *quantile*
After that use partition around We then recurse on each of the size. beginning with n
Since we get the $\left\lfloor \frac{k}{2} \right\rfloor$*, this is in the position* $\left\lfloor \frac{n}{2} \right\rfloor$ *. we will then recurse on arrays*
$A\left[r: \left\lfloor \frac{n}{2} \right\rfloor - 1\right]$ *and* $A\left[\left\lfloor \frac{n}{2} \right\rfloor + 1: p\right]$ *and we will find the k − th quantiles of them*
in the same. this can be done because we selected and removed thek − th quantile
this means that $\frac{n}{k}$ *positions ahead and below there are k − th quantiles*
All of this considering that k is even, now if k is odd we will get still the $\left\lfloor \frac{k}{2} \right\rfloor$
In this situation we will divide the recursion on $\left\lfloor \frac{k}{2} \right\rfloor \cdot \left\lfloor \frac{n}{k} \right\rfloor$.
Based on that we can have the following recursion
$T(n) = T\left(\left\lfloor \frac{k}{2} \right\rfloor \cdot \left\lfloor \frac{n}{k} \right\rfloor\right) + T\left(\left\lceil \frac{k}{2} \right\rceil \cdot \left\lfloor \frac{n}{k} \right\rfloor\right) + \Theta(n)$
We continue calling until n > k. In first level we do n, in the second we do n − 1 work
Because we just remove one pivot and we iterate over the remaining arrays
We know we discover one k − th qauntile per iteration so the number of nodes in
our tree must be k. The height of our tree is $\log_2 k + 1$ *we end with the following sum*

$$\sum_{i=1}^{\log_2 k+1} n - (i-1) \rightarrow \sum_{i=1}^{\log_2 k} n - i \rightarrow nlog_2(k) - \frac{(\log_2 k - 1)\log_2 k}{2} \rightarrow k < n$$

What lead our algorithm to be nlog(k)

9.3.9 *Describe an O(n) − time algorithm that, given a set S of n distinct numbers and a positive integer k ≤ n, determines the k numbers in S that are the closest to the median of S*

We first need to select the median, we can use the SELECT algorithm to do it.
We can them create an array of such that $A = \{|x - y|; y \in S\}$ *x is the median.*
Now we have an array with the values of difference. we can maintain a correspondence using a map
the elements with the minimun amount are the closest. based on that we can use
select to get the k − statistical order. With that we can Partition it. So we endup
with the k − numbers in the left and associated with the real closest numbers

---

9.3.10 *Let X[1: n]and Y[1: n]be two arrays, each containing n numbers already in sorted order. Give an O(log n) − time algorithm to find the median of all 2n elements in arrays X and Y. Assume that all 2n numbers are distinct*

Lets analyze the two medians, one from each set, let denote the sets by A and B
the median index of each set by $\left\lfloor\frac{n}{2}\right\rfloor = m$, *so* A_m *and* B_m *are the medians of each set*
and A_i *and* B_i *are the elements in position i of the sets A and B*
Lets com A_m *with* B_m. *if* $A_m < B_m \rightarrow m$ *here is a special index,*
maybe the median is not a member of the array, if n is even. What we know is that
A_m *and* B_m *are greater than* $\left\lfloor\frac{n}{2}\right\rfloor$ *elements each and lower than* $\left\lfloor\frac{n}{2}\right\rfloor$.
if $A_m < B_m \rightarrow B_m$ *is greater than* $2 \cdot \left\lfloor\frac{n}{2}\right\rfloor$. *Lets suppose here that n is even,*
so B_m *is greater than n elements* $= \frac{2n}{2}$. *it can be a median. Since n is even*
the element $A_{\left\lceil\frac{n}{2}\right\rceil}$ *is still greater than n elements, the ones that exist in the array*
The median of 2n elements must be greater exactly of n members. This means that all elements
above him are not suitable to be the median, since they would be greater than more tha n elements
The same analysis could be done with A_m *it means it is smaller than n elements*
Since it doesn'texist it means that $A_{\left\lfloor\frac{n}{2}\right\rfloor}$ *is lower than n elements and could be the median*
But all others below it cannot. So we reduce from 2n to n potential elements in each comparison
of the medians, since we are removing from the ordered arrays, they are still ordered
We just need to adjust the index of the new median, what takes constant time
Since n is even we need to take 2n to 2 elements, since we reduce by half in each step
We can take $O(\lg(n))$ *as the height, that is the time complexity of the algorithm.*
Same analysis go for n odd. We just have a problem when we have 2 elements in each array
In this situation we cannot exclude non possible medians, but we can find the medians
in this situation in constant time, what leads the correct result

---

9.1 Largest i numbers in sorted order.
You are given a set of n numbers, and you with to find the i largest in sorted order
using a comparison − based algorithm. Describe the algorithm that
implements each of the following methods with the best asymptotic worst
−case running time, and analyze the running times of the algorithms in terms
of n and i
a) Sort the numbers, and list the i largest number.
b)Build a max − priority queue from the numbers, and call EXTARCT − MAX
i times.
c) Use an order − static algorithm to find the i − th largest number
, partition around that number, and sort the i largest numbers

a) It was prove that by a decision tree that the best we can do in a comparison
−based algorithm to sort is $\Omega(\text{nlogn})$ *in the worst case. the i − largest is simply*
the i − th elements after sorted, so the running time is $\Theta(nlgn)$

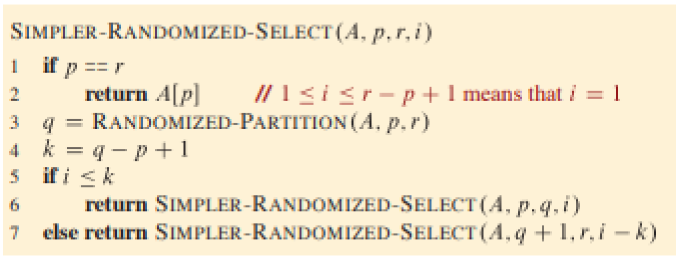
b) Using the heap structure to implement a priority queue we know that
it takes $\Theta(n)$ *time. Since we want the i − largest. we extract the first, which is*
constant, we do a max − heapify, which takes $\log(n)$ *and repeat. we have to do i − 1*
max − heapifies, or i if we want to maintain the data structure

In this situation, the total ammount of work done here is $\sum_{j=1}^{i} \log(n-j) + \Theta(n)$

$\log\left(\frac{n!}{(n-i-1)!}\right) + \Theta(n)$. *which leads us to* $\Theta(i \cdot log(n) + n)$

c) select and parition takes $\Theta(n)$. *Now we need to sort the* $i-th$ *largest*
it have i elements so we have $\Theta(n + i \cdot \lg(i))$

9.2 *Variant of randomized selection*
Professor Mendel has proposed a simplifying RANDOMIZED − SELECT
by eliminating the check for wheter i and k are equal. The simplified procedure is SIMPLER − RANDOMIZED − SELECT



```
SIMPLER-RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]        // 1 ≤ i ≤ r − p + 1 means that i = 1
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i ≤ k
    return SIMPLER-RANDOMIZED-SELECT(A, p, q, i)
else return SIMPLER-RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

A) Argue tht in the worst case. SIMPLE − RANDOMIZED − SELECT never terminates

The idea ofthe argument of the algorithm is simple. We partition our array
based on some value, this partition have value q and p have the index of the first element
What the k is doing is calculate how many elements there are in the first side
Based on that we verify if the value we want to calculate is there. If that's the case
We recurse on this side of the array. if it's not we call it on the other side of the array
Now lets a situation where $p = 1$ *and* $r = 10$. *if we get r as the pivot, the greater*
element, then $q = 10$ *and* $k = 10$ *i is always less than 10. We will then recurse on the*
same input array

b)Prove that the espected running time of SIMPLER − RANDOMIZED − SELECT is still $\mathbf{O}(\boldsymbol{n})$

We basically have a $T(n)$*algorithm and in this situation we basically use Pivot to*
to reduze the size of n, and we recurse on a new i.
$T(n, i) = T(n-k, g(i)) + \Theta(n)$, *where* $0 \le k \le n-1$
$n-k$ *is in fact a random variable X, that represents the resulting size of the array*
the possible values for X is $1,2,3 \dots n-1$. *What is the probability of each one of them*
? It depends on i and. there is two situation where we end up with 1, there is two situations
where we end up with size 2 and so on. there is only one case where there is only one situation
thats when n is odd and we choose the median as the pivot, but since we have two
divisions of same length, it still counts for two cases, but only one pivot produces it
we have $n-1$ *possible sizes each one with the same probability.*
So what is the expected size of this new array? $\frac{1}{n-1}$ *is the probability of a size*
being chosen. $\frac{1}{n-1}\sum_{i=1}^{n-1} i = n \cdot \frac{n-1}{2} \to \frac{n \cdot n - 1}{2 \cdot (n-1)} \to \frac{n}{2}$ *is the expected size.*
$E(T(n)) = E(T(n-k)) + \Theta(n) = T(E(n-k)) + \Theta(n) \to$
$E(T(n)) = T\left(\frac{n}{2}\right) + \Theta(n)$
This recurrence have solution $O(n)$.

9.3 Weigheted Median

Consider n elements $x_1, x_2 \dots x_n$, with positive weighs $w_1, w_2 \dots w_n$ such that $\sum_{i=1}^{n} w_i = 1$. The weighted(lower)median is an element x_k satisfying:

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

For example, consider the following elements x_i and weights w_i:

i	1	2	3	4	5	6	7
x_i	3	8	2	5	4	1	6
w_i	0.12	0.35	0.025	0.08	0.15	0.075	0.2

For those elements, the median is $x_5 = 4$, but he weighted median is $x_7 = 6$.
To see why the weighted median is x_7, observe that the elements less than x_7 are x_1, x_3, x_4, x_5 and x_6, and the sum $w_1 + w_3 + w_4 + w_5 + w_6 = 0.45$, which is less than $\frac{1}{2}$. Furthermore, only element x_2 is greater than x_7 and $w_2 = 0.35$, which is no greater than $\frac{1}{2}$

a) Argue that the median of $x_1, x_2 \dots x_n$ is the weigheted median of the x_i with the weights $\frac{1}{n}$ for $i = 1, 2 \dots n$

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

The post − office location provlem is defined as follows. The input is n points $p_1, p_2 \dots p_n$ with associated weights $w_1, w_2 \dots w_n$. A solution is a point p(not necessarily one of the input points) that minimized the sum $\sum_{i=1}^{n} w_i d(p, p_i)$, where $d(a, b)$ is the distance between points a and b.

d) 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 the 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 iven by $|x_1 - x_2| + |y_1 - y_2|$

a) We know that the weighted median(lower)is defined as x_k where $\sum_{x_i<x_k} w_i < \frac{1}{2}$ and $\sum_{x_i>x_k} w_i \le \frac{1}{2}$, since w_i is $\frac{1}{n}$ we have that $\frac{k}{n} < \frac{1}{2}$ and $\frac{j}{n} \le \frac{1}{2}$. Let x be the median. n being odd

$x\ have\ \frac{(n-1)}{2}\ as\ k\ and\ j\ and\ this\ values\ as\ j\ and\ n\ assures\ that$
$the\ equations\ above\ holds, so\ the\ median\ x\ is\ also\ the\ weighted\ median$

$b)\ Lets\ sort\ the\ array, it\ takes\ O\big(nlg(n)\big).\ We\ have\ associated\ with\ it$
$the\ weights\ of\ each\ value. For\ a\ value\ in\ the\ position\ k\ to\ be\ the\ Weighted$
$median, we\ need\ that\ all\ values\ lower\ than\ k\ have\ a\ sum\ of\ the\ index\ until\ it$
$less\ than \frac{1}{2}. Based\ on\ that\ we\ can\ iterate\ over\ these\ weights\ and\ create$
$an\ array\ A\ in\ \Theta(n) time\ in\ each\ index\ 1\ contains\ the\ sum\ of\ weights\ from\ 1\ to\ 1$
$index\ 2\ contains\ the\ sum\ of\ weights\ from\ 1\ to\ 2\ and\ so\ on.$
$Now\ we\ do\ something\ similar\ but\ in\ the\ other\ direction, we\ create\ a\ array\ B$
$In\ which\ the\ value\ in\ the\ index\ i\ is\ equal\ the\ sum\ of\ all\ weights\ from\ i+1\ to\ n$
$if\ we\ begin\ from\ the\ end\ we\ still\ run\ this\ program\ in\ \Theta(n). The\ array\ B, now\ indicates$
$from\ a\ given\ element\ the\ sum\ of\ weights\ of\ the\ elements\ greater\ or\ equal\ than\ it.$
$We\ just\ need\ to\ find\ the\ position\ that\ in\ both\ arrays\ obey\ the\ rules\ for\ the\ weigheted\ lower\ median$
$which\ takes\ more\ \Theta(n).$

$c) If\ we\ use\ the\ select\ to\ find\ the\ median\ and\ the\ partition\ around$
$We\ can\ verify\ in\ \Theta(n) what\ is\ the\ value\ of\ the\ sum\ of\ weights\ of\ the\ elements\ lower\ than$
$the\ median\ and\ the\ sum\ of\ elements\ that\ are\ greater\ than\ the\ median.$
$If\ the\ sum\ of\ elements\ greater\ are\ higher\ than \frac{1}{2}\ the\ median\ is\ not\ the\ right\ choice. but$
$more\ than\ that\ the\ half\ part\ above\ it\ its\ not\ the\ weighted\ median\ because\ the\ sum$
$of\ the\ elements\ lower\ than\ them\ is\ the\ same\ as\ the\ median + the\ weight$
$of\ the\ median\ which\ is\ clearly > \frac{1}{2}\ . we\ can\ then\ reduce\ the\ size\ of\ the$
$array\ by\ half\ and\ recurse\ what\ would\ leads\ to\ something\ like\ T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$
$Now\ if < \frac{1}{2}\ we\ analyze\ th\ second\ case, if > \frac{1}{2}\ we\ have\ the\ same\ situation, but\ for\ the$
$lower\ \ first\ half\ so\ we\ can\ eliminate\ them\ and\ recurse, if\ it\ is \leq \frac{1}{2}\ we\ found\ it!$
$So\ in\ the\ worst\ case\ we\ recurse\ until\ n = 1, or\ something\ like\ that.$
$In\ the\ worst\ case\ we\ end\ up\ with\ the\ recurrence\ T(n) = T\left(\frac{n}{2}\right) + \Theta(n) which\ is\ \Theta(n)$

$d) When\ the\ derivative\ is\ negative\ our\ function\ values\ are\ decrementing, when$
$the\ derivative\ is\ positive\ our\ function\ values\ are\ incrementing.$
$Based\ on\ that\ given\ a\ value\ p_x\ is\ such\ that\ f'(p) < 0 for\ p < p_x\ and\ f'(p) > 0$
$for\ p > p_x\ we\ have\ that\ p_x\ is\ a\ point\ of\ minimun. Based\ on\ that$
$To\ minimize\ the\ sum\ \sum_{i=1}^{n} w_i d(p, p_i)\ \ we\ can\ we\ realize\ the\ derivative, Basically$
$We\ endup\ with:$
$f'(p) = \sum_{p_i \leq p} w_i - \sum_{p_i > p} w_i \rightarrow We\ want\ a\ point\ that\ for\ the\ an\ element\ in\ the\ set\ p_i$
$f'^{(p_k)} \leq 0\ and\ f'^{(p_{k+1})} > 0. In\ this\ region\ we\ have\ a\ point\ of\ minimun$
$For\ a\ given\ p_k \rightarrow$
$$\sum_{p_k \leq p} w_i - \sum_{p_k > p} w_i \leq 0\ and\ \sum_{p_{k+1} \leq p} w_i - \sum_{p_{k+1} > p} w_i > 0 \rightarrow$$

$$\sum_{p_k \le p} w_i + \sum_{p_k > p} w_i \le 2 \sum_{p_k > p} w_i \ and \ \sum_{p_{k+1} \le p} w_i + \sum_{p_{k+1} > p} w_i > 2 \sum_{p_{k+1} > p} w_i \rightarrow$$
$$1 \le 2 \sum_{p_k > p} w_i \rightarrow 1 > 2 \sum_{p_{k+1} > p} w_i \rightarrow$$
$$\sum_{p_{k+1} > p} w_i < \frac{1}{2} \ and \ \sum_{p_k > p} w_i \ge \frac{1}{2} \rightarrow \sum_{p < p_{k+1}} w_i < \frac{1}{2} \ and \ \sum_{p > p_k} w_i < \frac{1}{2}$$
Now for any p between $p_k < p < p_{k+1}$ *the equation above are valid*
$$If \sum_{p < p_{k+1}} w_i < \frac{1}{2} \ and \ p_k < x_k < p_{k+1} \rightarrow \sum_{p < x_k} w_i < \frac{1}{2}$$
$$if \sum_{p > p_k} w_i < \frac{1}{2} \ \ and \ \ p_k < x_k < p_{k+1} \rightarrow \sum_{p > x_k} w_i < \frac{1}{2}$$

e) In this case we have just two problems that were derived above.
We have that the manhattan distance is given by the following:
$d(a,b) = |x_1 - x_2| + |y_1 - y_2|$ *Now our function becomes:*
$$f(\vec{p}) \sum_{i=1}^{n} w_i \cdot (|x - x_2| + |y - y_2|) = f(x,y) = \sum_{i=1}^{n} w_i \cdot |x - x_2| + \sum_{i=1}^{n} w_i \cdot |y - y_2| \rightarrow$$
x and y are totally independent so $f(x,y) = g(x) + h(y)$*, this means that the minimun of* $f(x,y)$ *is attainable at the minimun of* $g(x)$ *and* $h(y)$ *and the minimun of* $g(x)$ *and* $f(y)$ *happens to be the weighted average as showed in the previous exercise. So The best solution is the weighted average for the x − coordinate and y − coordinate*

9.4 *Small order statistics*

Lets denote by $S(n)$ ***the worst case number of comparisons used by SELECT to select the i − th order statistic from n numbers. Although*** $S(n) = \Theta(n)$***, the constant hidden by the*** Θ **− notation is rather large. When i is small relative to n, there is an algorithm that uses SELECT as subroutine but makes fewer comparisons in the** ***worst case.***

a) Describe an algorithm that uses $U_i(n)$ ***comparisons to find the ith smallest of n elements where***
$$U_i(n) = f(x) = \begin{cases} S(n), & if\ i \le \frac{n}{2} \\ \left\lfloor \frac{n}{2} \right\rfloor + U_i\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + S(2i), & otherwise \end{cases}$$

(Hint: Begin with $\left\lfloor \frac{n}{2} \right\rfloor$ ***disjoint pairwise comparisons, and recurse on the set containing the smaller element from each pair)***

b) Show that, if $i < \frac{n}{2}$ ***. then*** $U_i(n) = n + O\left(S(2i) lg\left(\frac{n}{i}\right)\right)$

c) Show that if i is a constant less than $\frac{n}{2}$ ***then*** $U_i(n) = n + O(lg(n))$

d) Show that if $i = \frac{n}{k}$ ***for*** $k \ge 2$ ***then*** $U_i(n) = n + O\left(S\left(\frac{2n}{k}\right) log(k)\right)$

a)Lets suppose initially that n is even and divide it in the middle.
Now we compare the first elements of each array. We them put the smaller

$of\ the\ comparison\ in\ the\ higher\ array\ part. We\ know\ that\ the\ SELECT\ algorithm$
$is\ such\ that\ it\ partitions\ the\ input\ array\ in\ a\ such\ way\ that\ the\ first\ i\ elements\ are\ the\ i$
$-th\ smallest\ elements. We\ then\ recurse\ this\ on\ the\ upper\ side, We\ do\ it\ in\ a\ way\ that\ guarantees$
$The\ upper\ side\ contains\ the\ i - smallest\ elements\ and\ the\ larger$
$counterparts\ that\ are\ placed\ in\ the\ first\ i\ elements\ of\ the\ array. We\ then\ Use\ select\ on\ this\ 2i$
$array, what\ results\ in\ the\ time\ complexity\ above$

$b) U_i(n) = \left\lfloor \frac{n}{2} \right\rfloor + U_i\left(\frac{n}{2}\right) + S(2i), lets\ use\ the\ substitution\ method\ to\ show\ that\ if\ i$
$< \frac{n}{2} then\ U_i(n) \le n + cS(2i)\lg\left(\frac{n}{i}\right) - d(loglogn)S(2i) \rightarrow$
$U_i(n) = \left\lfloor \frac{n}{2} \right\rfloor + U_i\left(\frac{n}{2}\right) + S(2i)$
$$\le \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil + cS(2i)\lg\left(\left\lceil \frac{n}{2} \right\rceil\right) - cS(2i)\lg(i) - d\left(lglg\left\lceil \frac{n}{2} \right\rceil\right)S(2i)$$
$\le n + cS(2i)\lg\left(\frac{n}{2}+1\right) - cS(2i)\lg(i) - d(\lg(\lg(n-1)))S(2i) \le$
$n + cS(2i)\lg n - cS(2i)\lg(i) - d(\lg(\lg(n)))S(2i) \rightarrow$
$this\ will\ hold\ if\ we\ guarantee\ that cS(2i)\lg\left(\frac{n}{2}+1\right) - d(\lg(\lg(n-1)))S(2i)$
$$\le cS(2i)\log(n) - dlg(\lg(n))S(2i) \rightarrow$$
$This\ happens\ if\ and\ only\ if\ \ clg\left(\frac{n}{2}+1\right) - d(\lg(lgn-1)) \le clg(n) - dlglgn$
$c\left(\lg\left(\frac{\frac{n}{2}+1}{n}\right)\right) \le dlg\left(\frac{lgn-1}{\lg(n)}\right) \rightarrow c\left(\lg\left(\frac{1}{2}+\frac{1}{n}\right)\right) \le dlg\left(\frac{lgn-1}{\lg(n)}\right)$
$The\ limit\ to\ infinity\ of\ the\ left\ goes\ to - c\ while\ the\ limit\ of\ the\ right\ goes\ to\ 0$
$The\ left\ equation\ is\ monotonically\ decreasing\ and\ the\ liimmt\ of\ the\ right\ is$
$monotonically\ increasing, for\ n = 4\ we\ need\ that\ clg\left(\frac{3}{4}\right) \le dlog\left(\frac{1}{2}\right) \rightarrow$
$When\ d \le clg\left(\frac{4}{3}\right) it\ holds\ for \ge 4\ so\ we\ are\ done$

$given\ that\ i\ is\ constant\ less\ than \frac{n}{2}\ we\ have\ that\ S(2i) is\ also\ a\ constant$
$ang\ \lg\left(\frac{n}{i}\right) = \log(n) - c\ which\ is\ O(\log(n)) so U_i = n + O(constant \cdot O(\log(n))) \rightarrow$
$U_i(n) = n + O(\lg(n))$

$d)\ We\ just\ need\ to\ replace\ it\ in\ the\ previous\ equations\ so\ we\ end\ up\ with\ the\ following:$

$U_i(n) = n + O\left(S(2i)\lg\left(\frac{n}{i}\right)\right) = n + O\left(S\left(\frac{2n}{k}\right)\lg(k)\right)$

$\boldsymbol{9.5\ alternative\ analysis\ of\ randomized\ selection}$
$\boldsymbol{In\ this\ problem, you\ will\ use\ indicator\ random\ variable\ to\ analyze\ the\ procude\ RANDOMIZED}$
$\boldsymbol{-SELECT\ in\ a\ manner\ akin\ to\ our\ analysis\ of\ RANDOMIZED - QUICKSORT}$
$\boldsymbol{in\ Section\ 7.4.2. As\ in\ the\ quicksort\ analysis, we\ assume\ that\ all\ elements\ are\ distinct\ and}$
$\boldsymbol{we\ rename\ the\ elements\ of\ the\ input\ array\ A\ as\ z_1, z_2 \ldots z_n\ where\ z_i\ is\ the\ i}$
$\boldsymbol{-th\ smallest\ element.\ Thus\ the\ call\ RANDOMIZED - SELECT(A, 1, N, i) returns}$
$\boldsymbol{z_i. For\ 1 \le j < k \le n, let}$
$\boldsymbol{X_{ijk}}$
$\boldsymbol{= I\{z_j\ is\ compared\ with\ z_k\ sometime\ during\ the\ execution\ of\ the\ algorithm\ to\ find\ z_i\}}$

$a)$ Given an exact expression for $E[X_{ijk}]$; (hint: Your expression may have different values, depending on the values of i, j and k)

$b)$ Let X_i denote the total number of comparisons between elements of array A when finding z_i. Show that:

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

$c)$ Show that $E[X_i] \leq 4n$

$d)$ Conclude that, assuming all elements of array A are distinct, RANDOMIZED SELECT runs in $O(n)$ expected time

a) We have an analysis similar to the one realized for the quicksort procedure
Based on the randomized
– select algorithm, comparisons are just realized in the scope of the RANDOMIZED
– PARTITION procedure, and we already verified that two elements z_k and z_j
Are just compared if one of them is choosen as pivot before the other elements
between them, moreover, they are just compared one time. there are other situation in which
maybe they are not compared, if $i < j$ and $i < k$ or $i > j$ and $i > k$.
in these situations maybe the elements are not even verified. We have then 3 situations that
we must analyze . Lets assume without loss of generality that $j \leq k$ then we have:
if $j \leq i \leq k$ the size of the set is $k - j + 1$, if $i \leq j \leq k$ then the size is $k - i + 1$
if $j \leq k \leq i$ the size is $i - j + 1$. Now the probability is that one of them is chosen
as pivot before the other, and again since they are likely to be chosen we end up with:

$$E[X_{ijk}] = f(x) = \begin{cases} 2/(k-j+1) \\ 2/(k-i+1) \\ 2/(i-j+1) \end{cases}$$

b)So X_{ijk} is the random variable responsible for denoting 1 if in the search for i we compare j and k
And we want the total number of comparisons.

$$\sum_{j=1}^{n-1}\sum_{k=j+1}^{n}\Pr(k,j) = \sum_{j=1}^{n-1}\sum_{k=j+1}^{n}\mathrm{E}\left[\mathrm{X_{ijk}}\right] \quad \textit{This is only one of the three,}$$

so its clearly bounded by the sum of all of them, so we know that our expectation is as follow:

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

Just realize arithmetic operations and we end up with the desired equation

$$c)\textit{We need to show that} \sum_{j=1}^{i}\sum_{k=i}^{n}\frac{1}{k-j+1}+\sum_{k=i+1}^{n}\frac{k-i-1}{k-i+1}+\sum_{j=1}^{i-2}\frac{i-j-1}{i-j+1} \leq 2n$$

For that we will calculate the maximun values of each summation and show that its less than $2n$
For the last summation we have a function like $\frac{k-1}{k+1}$ which is incrasing, so its maximun
value happens for the higher value possible for k, in this case the higher value for k
$i - j$ happens j is the lowest as possible and i the largest as possible:

$$\frac{n-1-1}{n-1+1} = \frac{n-2}{n} \; so \sum_{j=1}^{i-2}\frac{i-j-1}{i-j+1} \leq \frac{n-2}{n}\cdot(n-2), \textit{for the second summation its the}$$

same logic, the maximun k is n and the minimun i is 1:

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

Besides it we kwow that each term is less than 1 so the total of the two summations are less than: $n-i-1+1+i-2=n-2$. Now we need to show that our first summation is lower than $n+2$.

We have the following sum $\sum_{j=1}^{i}\sum_{k=i}^{n}\frac{1}{k-j+1} \leq \sum_{j=1}^{i}\sum_{k=i}^{n}\frac{1}{i-j+1} = \sum_{j=1}^{i}\frac{(n-i+1)}{i-j+1} =$

$(n-i+1)\sum_{j=1}^{i}\frac{1}{j}$ this is always less then $n+2$

d) We can reuse the lemma statede in the Quicksort section, that the running time is $O(n+X)$, where X is the number of comparisons

---