7.1.1 *Using figure 7.1 as a model, illustrate the operation of PARTITION on the array*: $A = 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11$

13 19 9 5 12 8 7 4 21 2 6 11
13 19 9 5 12 8 7 4 21 2 6 11
13 19 9 5 12 8 7 4 21 2 6 11
9 19 13 5 12 8 7 4 21 2 6 11
9 5 13 19 12 8 7 4 21 2 6 11
9 5 13 19 12 8 7 4 21 2 6 11
9 5 8 19 12 13 7 4 21 2 6 11
9 5 8 7 12 13 19 4 21 2 6 11
9 5 8 7 4 13 19 12 21 2 6 11
9 5 8 7 4 13 19 12 21 2 6 11
9 5 8 7 4 2 19 12 21 13 6 11
9 5 8 7 4 2 6 12 21 13 19 11
9 5 8 7 4 2 6 11 21 13 19 12

7.1.2 *What value of q does PARTITION return when all elements in the subarray $A[p:r]$ have the same value? Modify Partition so that $q = \left\lfloor \frac{p+r}{2} \right\rfloor$*

Since all have the same value we will aways increment j and i. i always have the same value as j in the end of the for loop, so when $j = r - 1$, i at the end is $r - 1$.
We then return r. So we have a totally unbalanced array with $q = r$
we can use a flag to indicate if the it was equal or less than, our code would be:

```
Partition(A, p, r)
x = A[r]
i = p − 1
balance = 0
for j = p to r − 1
  if A[j] < x:
    i = i + 1
    exchange A[i] with [j]
else if A[j] == x:
   if balance < 0:
      i = i + 1
      exchange A[i] with [j]
      balance = balance + 1
   else:
     balance = balance − 1
```

In this way when we find an element equal we decide to put it as lower, *or in the left side*
But if we find another, we put it as higher *and put it to the right side*

7.1.3 *Give a Brief argument that the running time of PATITION on a subarray of size n is $\Theta(n)$*

We do constant work on each iteration of for loop. The size of for loop is $r - 1 - p + 1 = size - 1$ iterations, if $size = n$ then we have $(n - 1) \cdot c$ work done which is $\Theta(n)$

7.1.4 *Modify QUICKSORT to sort into monotonically decreasing order*

We just need to swap the condition in the line 4 of the PARTITION subroutine
We change the line 4 to $if\ A[j] \geq x$

7.2.1 *Use the substitution method to prove that the recurrence* $T(n) = T(n-1) + \Theta(n)$ *has the solution* $T(n) = \Theta(n^2)$

$\Theta(n)$ *is a function such that* $\exists c_1, c_2$ *and* $n_0 > 0$ *such that* $0 \leq c_1 n \leq \Theta(n) \leq c_2 n\ \forall n \geq n_0$
Suppose that $T(k)$*is a function such that* $c_3 n^2 \leq T(k) \leq c_4 n^2$ $for\ n_0 \leq k < n$
We then have $T(n) = T(n-1)\Theta(n) \rightarrow$
$c_3(n-1)^2 + c_1 n \leq T(n) \leq c_4(n-1)^2 + c_2 n\ (if\ n-1 \geq n_0 \rightarrow if\ n \geq n_0 + 1)$
$c_3 n^2 - 2c_3 n + c_1 n + c_3 \leq T(n) \leq c_4 n^2 - 2c_4 n + c_2 n + c_4 \rightarrow$
$if\ -2c_3 n + c_1 n + 1 > 0\ and\ -2c_4 n + c_2 n + 1 < 0\ we\ have$:
$c_3 n^2 \leq T(n) \leq c_4 n^2$
$n(c_1 - 2c_3) + c_3 \geq 0$ *, just get* $c_3 = \frac{c_1}{2}$
$n(c_2 - 2c_4) + c_4 \leq 0\ if\ c_4 = c_2$ *then we have* $c_2(1-n)$
≤ 0. *we just need* n_0*tobe one and we are done*
we now need to verify the case $for\ n_0 \leq n < n_0 + 1$. So just the n_0
$T(n_0) = constant, since\ its, by\ assumption, an\ algorithmic\ recursion\ we\ are\ done$

7.2.2 *What is the running time of QuickSort when all elements of Array A have the same value*?

$\Theta(n^2)$. *Its the exaclty same recurrence as above*

7.2.3 *Show that the running time of QUICKSORT is* $\Theta(n^2)$ *when the array A contains distinct elements and is sorted in decreasing order*.

Since its in decreasing order we have $A_j < A_i\ for\ all\ j > i$. *since the pivot is the last, the pivot is lower than everyone. In this case, during the the inner loop, every comparison is false and so the pivot is exchanged with the first element. This makes the subarray* $A[P+1:r]$*be in decreasing order, except for the last element that now is the largest In the following loop, every iteration will lead the comparison to be true. since the first one What does it imply? imply that in the swapping step i will always be equal j. In this situation The order of the elements effectively doesn't change. In the last step the pivot will remain in this position and will be extracted. In the next iteration it's clear that the pivot is now the lowest element again, repeating the cycle. We could prove it by induction*

7.2.4 ***Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time − of − transaction ordering to check − number ordering is therefore the problem of sorting almost − sorted input. Explain persuasively why the procedure I NSERTION − SORT might tend to beat the procedure QUICKSORT on this problem.***

In the best case Quicksort is nlgn, while insertion sort is $\Theta(n)$. *Since the input is almost sorted And the insertion sort is based on considering that the array* $A\{0:i\}$*is already sorted it means that on average there is a* k*, in which each check is distant on average, a contant number* k *from its correct order. So Insertion sort will do on average* k *swaps per element since it iterates over n elements we get a average of* $\Theta(n)$

7.2.5 ***Suppose that the splits at every level of quicksort are in the constant proportion*** α ***to*** β***, where*** $\alpha + \beta = 1$ ***and*** $0 < \alpha \leq \beta < 1$***. Show that the minimum depth of a leaf in the recursion tree is approximately*** $\log_{1/\alpha}$ ***(Don't worry about integer round − off)***

For each level of quicksort applied on an array $A[p:r]$*we have a total amout of* $r - p$ *elements that will receive a new quicksort, since the size is* $r - p + 1$ *and we take out the pivot. We will divide it into two parts* $\rightarrow T(n) = T\big(\alpha(n-1)\big) + T\big(\beta(n-1)\big) + \Theta(n)$

At the second level we have the following:

$T\big(\alpha(n-1)\big) = T\big(\alpha(\alpha(n-1)-1)\big) + T\big(\beta(\alpha(n-1)-1)\big) + \Theta\big(\alpha(n-1)\big)$

$T\big(\beta(n-1)\big) = T\big(\beta(\alpha(n-1)-1)\big) + T\big(\beta(\beta(n-1)-1)\big) + \Theta\big(\beta(n-1)\big)$

we have the following guess of formula*looking always to only* α *or only* β *side*:

$n \rightarrow \alpha n - \alpha \rightarrow \alpha^2 n - \alpha^2 - \alpha \rightarrow \alpha^3 n - \alpha^3 - \alpha^2 - \alpha \ldots \rightarrow$ *at the* k *level* $\alpha^{k-1} n - \sum_{i=1}^{k-1} a^i \rightarrow$

the same goes for $\beta \rightarrow$ *at the* j *level we have* $\beta^{j-1} n - \sum_{i=1}^{j-1} \beta^i$

We could prove it inductively on j *and* k*, but this will be ommited. Now lets find* k *and* j *such that the value converges to a low constant value, the base case. Lets consider it to be* 1

$$\alpha^{k-1} n - \sum_{i=1}^{k-1} a^i = 1 \rightarrow a^{k-1} n - \frac{a^k - a}{a-1} = 1 \rightarrow \frac{a^k n - a^{k-1} n - a^k + a}{a-1} = 1$$

$$a^k n - a^{k-1} n - a^k + a = a - 1 \rightarrow a^k n - a^{k-1} n - a^k = -1 \rightarrow$$

$$a^k \left(1 - n + \frac{n}{a}\right) = 1 \rightarrow a^k = \left(1 - n + \frac{n}{a}\right)^{-1} \rightarrow \left(a^k\right)^{-1} = \left(\left(1 - n + \frac{n}{a}\right)^{-1}\right)^{-1} \rightarrow$$

$$\left(\frac{1}{a}\right)^k = \left(1 - n + \frac{n}{a}\right) \rightarrow k = \log_{\frac{1}{a}}\left(1 - n + \frac{n}{a}\right).$$

The same goes for b. in a average case the $\log_{\frac{1}{a}}\left(1 - n + \frac{n}{a}\right) \sim \log_{\frac{1}{a}}(n)$

7.2.6 ***Consider an array with distinct elements and for which all permutations***

of the elements are eqaully likely. Argue that for any constant $\alpha \leq \frac{1}{2}$ ***the probability is approximately*** $1-2\alpha$ ***that Partition produces a split at least as balanced as*** $1-\alpha$ ***to*** α

Ok, so in this situation we won't consider the remove of Pivot
Since all elements are we receive an input as a sequence $Z = \{Z_1 \dots Z_n\}$
This sequence is such that, for an element Z_k *we have* $k-1$ *elements smaller and* $n-k$ *elements larger. we want that* $\frac{\max\{n-k, k-1\}}{\min\{n-k, k-1\}} \leq \frac{1-\alpha}{\alpha}$,
So we want that $\frac{n-k}{k-1} \leq \frac{1-\alpha}{\alpha}$ *and* $\frac{k-1}{n-k} \leq \frac{1-a}{a} \rightarrow$
$an - ak \leq k - ka - 1 + a \rightarrow an \leq k - 1 + a \rightarrow k \geq an - a + 1$
$n - na - k + ka \geq ak - a \rightarrow k \leq a + n - na$
$an - a + 1 \leq k \leq a + n - na$. *we know that k is an integer.* $\rightarrow$
so $1 \leq k - an + a \leq a + n - na - na + a \rightarrow$
$1 \leq k - an + a \leq 2a + n - 2na \rightarrow$ *so there is approximately* $2a + n - 2na$ *such* $Z_k's$ *that guarantees it. So since the chosen of one of them is equal, we have*:
$Pr = \frac{2a + n - 2na}{n} = \frac{2\alpha}{n} + 1 - 2\alpha \rightarrow 1 - 2\alpha \leq Pr \leq \frac{1}{n} + 1 - 2\alpha$
If n is large enough we are okay

7.3.1 ***Why do we analyze the expcted running time of a randomized algorithm and not its worst*** $-$ ***case running time***?
The worst case running time is exactly the same for the base Quicksort.
What the randomization generates is make the worst case less possible *for all type of inputs we receive.*

7.3.2 ***When Randomized*** $-$ ***QUICKSORT runs, how many calls are made to the random number generator RANDOM in the worst case? How about in the best case? Given your answer ins terms of*** Θ $-$ ***notation***

Both clearly are $\Theta(n)$

7.4.1 ***Show that the recurrence*** $T(n) = \max\{T(q) + T(n-q-1): 0 \leq q \leq n-1\} + \Theta(n)$ ***has a lower bound of*** $\Omega(n^2)$

Suppose that $ck^2 \leq T(k)$ *for* $n_0 \leq k < n$ *then* $\rightarrow$
$\max\{cq^2 + c(n-q-1)^2: 0 \leq q \leq n-1\} + c_1 n \leq T(n) \rightarrow$
$c\max\{q^2 + (n-q-1)^2: 0 \leq q \leq n-1\} + c_1 n \leq T(n) \rightarrow$
$cmax\{n^2 - 2n + 1 + 2q(q-(n-1)): 0 \leq q \leq n-1\} + c_1 n \leq T(n)$
$cmax\{n^2 - 2n + 1 + 2q(q-(n-1)): 0 \leq q \leq n-1\} \leq T(n)$
Now lets analyze $n^2 - 2n + 1 + 2q(q-(n-1))$ *of* $r \;\; 0 \leq q < n-1$
it has positive concavity and the minimun of the function occurs in $\frac{n-1}{2}$
Which is obviously lower than $n-1$, *so we just need to analyze* $q = 0$ *and* $q = n-1$
for $q = 0$ *we have* $(n-1)^2$ *and for* $q = n-1$ *we have* $(n-1)^2$ *so this is the max*
$c(n-1)^2 + c_1 n \leq T(n) \rightarrow cn^2 - 2cn + 1 + c_1 n \leq T(n) \rightarrow$

$cn^2 - 2cn + c_1 n \leq T(n) \rightarrow cn^2 + n(c_1 - 2c) \leq T(n).$ *we just need* $c_1 - 2c > 0 \rightarrow c < \frac{c_1}{2}$
So we have $cn^2 \leq T(n)$. *We would neet to analyze the* n_0 *for the* $\Theta(n)$*function and adjust with the minimun analyzes again, but this will be ommited*

7.4.2 *Show that quicksort's best case runing time is* $\Omega(n\,lgn)$

We will show that the best case running time is $\Omega(nlgn)$
So in every recursive call what we do is the same anaylsys as the previous execise, except that now we want the min, so we end up with the best case:
$T(n) = \min\{T(q) + T(n-1-q) : 0 \leq q \leq n-1\} + \Theta(n)$
Based on what we want to prove we will guess that $cklgk \leq T(k) for\ n_0 \leq k < n$
In this situation we have $\min\{qlg(q) + (n-1-q)\lg(n-1-q)\} + \Theta(n) \leq T(k)$
Lets analyze the function $qlg(q) + (n-1-q)\lg(n-1-q)$ *in relation to* q, *given* n
we have the derivative equals $\log(q) - \log(n-1-q) \rightarrow$
So lets analyze now the signal of this value. This is greater than 0 *if and only if* $q > n-1-q \rightarrow 2q > n-1 \rightarrow q > \frac{n-1}{2}$. *So untl* $\frac{n-1}{2}$ *the function is decreasing*
Therefore the minimun value happens when $q = \frac{n-1}{2}$. *This value is less then the min since the* $\min$ *can only attain integers so we have*:
$(n-1)\log\frac{(n-1)}{2} + \Theta(n) \leq T(n).$
$(n-1)\log(n-1) - n + \Theta(n) \leq (n-1)\log(n-1) - n + 1 + \Theta(n) \leq T(n)$
now we know that $\Theta(n)$ *is a function such exists*
c_1 *such that* $\Theta(n) \leq c_1 n\ \forall n \geq n_0$ *and we know that we can get this* c_1 *to be* > 1
We want the following to occur, we want to choose this c_1 *in a proper way that*
$nlg(n) \leq (n-1)\log(n-1) - n + \Theta(n) \rightarrow$
$nlg(n) - nlg(n-1) + \lg(n-1) + n \leq c_1 n \rightarrow$
$\lg\left(\frac{n}{n-1}\right) + \frac{1}{n}\lg(n-1) + 1 \leq c_1$. *If the left side is limited then we can choose such* c_1
This c_1 *will be the maximunbetween this upper bound for the left side of the equation above and the one that guarantees the properly definition of* $\Theta(n)$
the left side is limited if $\lg\left(\frac{n}{n-1}\right) + \frac{1}{n}\lg(n-1)$ *is limited. Now we assume that* $n > 1$. *The derivative of this function is* $-\frac{\ln(n-1)}{\ln(2)\,n^2}$ *which is always negative based on that we can limit, that is zero. and the maximun number occurs for* $n = 2$ *that is* 1; *So we can take the propers constant that all will hold*

7.4.3 *Show thaat the expression* $q^2 + (n-q-1)^2$ *achieves its maximun value over* $q = 0, 1, 2 \ldots n-1$ *when* $q = 0$ *or* $q = n-1$

We already did it previously this have a positive concavity, so it has a minimun
This point of minimun occur when the derivative is equal to $0 \rightarrow$ *this happens when* $2q + 1 - n = 0 \rightarrow \frac{n-1}{2}$. *This means that after that the function increases and previously the function only decreases, this means that* $q = 0$ *and* $q = n-1$ *are the higher values in the interval*

7.4.4 ***Show that RANDOMIZED − QUICKSORT expected running time is*** $\boldsymbol{\Omega(nlg(n))}$

From the already known expected formula we have that

$$E[QuickSortRandom] = \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k+1} \geq \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{2k} = \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{1}{k}$$

By the appendix of this book we have that

$$\ln(n-i+1) \leq \sum_{k=1}^{n-i}\frac{1}{k} \rightarrow \sum_{i=1}^{n-1}\ln(n-i+1) = \ln(n!) = \frac{\lg(n!)}{\lg(e)} = \Theta(\text{nlg(n)})$$

So we have proven that the expcted running time is $\Omega(nlgn)$

7.4.5 Coarsening the recursion, as we did in Problem 2 − 1 for merge sort, is a common way to improve the running time of quicksort in practice. We modify the base case of the recursion so that if the array has fewer than k elements, the subarray is sorted by insertion sort, rather than by continued recursive calls to quicksort. Argue that the randomized version of this sorting algorithm runs in $\mathbf{O\left(nk + nlg\left(\frac{n}{k}\right)\right)}$ ***expected time. How should you pick k, both in theory and in practice*****?**

Okay so we are doing a quicksort from n to k and after k we will provide an insertion sort algorithm. now we will call insertion sort on a k array.

The averase case for insertion sort is k^2*. and the expected number of leaves is* $\frac{n}{k}$*.*

The recursion stops at the level $\lg(n) - \lg k$ *In this way we get to* $\left(nk + nlg\left(\frac{n}{k}\right)\right)$

we want to minimize it. In theory we could find the minimun value of the function inside the O notation and taking the derivative. In fact we want that the expcted value of the based quicksort to be better than this, what would leads us to find a k such that $nlgn \geq nk + nlgn - nlgk \rightarrow lgk \geq k$*, what is not reasonable.*

We can add constant factors to it that multiply these, like:

$c_1 nlgn \geq c_2 nk + c_1 nlg(n) - c_1 nlgk \rightarrow lgk$

$> \frac{c_2 k}{c_1}$*. So apparently there is a possible situation that this works. In practice just try possible* $K's$ *and analyze the expected time of them*

7.4.6 ***Consider modifying the PARTITION procedure by randomly picking three elements from subarray*** $\boldsymbol{A\{p:r]}$ ***and partitioning about their median (the middle value of the three elements). Aproximate the proability of getting worse than an*** $\boldsymbol{\alpha - to - (1-\alpha)}$***split, as a function of*** $\boldsymbol{\alpha}$ ***in the range*** $\mathbf{0 < \alpha < \frac{1}{2}}$

A worse than $\alpha - to - (1-\alpha)$*split are the ones* β *to* $1-\beta$ *where* $\beta > \alpha$

In this situation to such a split occur we need to choose an element such that it's greater than

αn *elements, for such being chosen we need to choose one lower than him and one greater than him and the number itself then for such number we need the following*:

$\frac{\alpha n}{n} \cdot \frac{1}{n} \cdot \frac{(1-\alpha)n}{n} = \frac{\alpha(1-\alpha)}{n}$. *for the next* $\frac{\alpha n+1}{n} \cdot \frac{1}{n} \cdot \frac{(1-\alpha)n-1}{n}$ *and so on*:

until it achieves a number k such that $(1-\alpha)n - k = 0 \rightarrow$

$$\sum_{i=0}^{(1-\alpha)n} \left(\frac{\alpha n+i}{n}\right)\left(\frac{1}{n}\right)\left(\frac{(1-\alpha)n-i}{n}\right) = \sum_{i=0}^{(1-\alpha)n} \left(\alpha+\frac{i}{n}\right)\left(\frac{1}{n}\right)\left(1-\alpha-\frac{i}{n}\right)$$

7.1 *Hoare partition correctness*

The version of Partition given in this chapter is not the original partitioning algorithm. Here is the original partitioing algorithm, which is due to C. A. R Hoare

```
HOARE-PARTITION(A, p, r)
1   x = A[p]
2   i = p − 1
3   j = r + 1
4   while TRUE
5       repeat
6           j = j − 1
7       until A[j] ≤ x
8       repeat
9           i = i + 1
10      until A[i] ≥ x
11      if i < j
12          exchange A[i] with A[j]
13      else return j
```

a) Demonstrate the operation of Hoare − Parition on the Array A $= 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21$

its just a basic operation, it will be ommited

b) Describe how the partition procedure in Section 7.1 differs from Hoare Partitionon when all elements are equal. Describe a practical advantage of Hoare − Partition over partition use in quicksort

Lets look at a practical example. Lets take the array $A = 1,1,1$

From the partition we have the following:

$x = A[3] = 1$

$i = 0$

for $j = 1$:

if $A[1] \leq 1$

$i = i + 1$

exchange $A[1]$*with* $A[1]$

For $j = 2$:

if $A[2] \leq 1$:

$i = i + 1$

exchange $A[2]$ *with* $A[2]$

exchange $A[3]$ *with* $A[3]$:

return 3

So we end up with $A = 1,1,1$

$Now\ taking\ a\ look\ at\ the\ Hoare-Partition\ method$:
$x = A[1] = 1$
$i = 0$
$j = 4$
$While\ true$:
$j = 3$
$i = 1$
$exchange\ A[1] with\ A[3]$
$A = 1,1,1$
$j = 2$
$i = 2$
$return\ 2$
$Hoare\ is\ typically\ better\ when\ there\ are\ duplicates, assuring\ a\ more\ balanced\ pivot$

The net three questions ask you to give a careful argument that the procedure HOARE − PARTITION is correct. Assuming that the subarray $A[p:r]$ contains at least two elements prove the following:

c) The indices i and j are such that the procedure never access an element of A outside the subarray $A[p:r]$

$In\ the\ first\ iteraction\ i\ and\ j\ are\ setted\ in\ a\ way\ to\ fall\ in\ the\ array\ bound$
$i\ is\ setted\ to\ p\ before\ acessing\ an\ array\ element\ and\ j\ is\ setted\ to\ r.$
$Since\ we\ will\ swap\ elements\ of\ index\ j\ only\ if\ A[j] is\ less\ then\ the\ pivot\ and\ A[i] is\ lower$
$Than\ the\ pivot\ we\ can\ guarantee\ that\ for\ all\ j, A[k] for\ j \leq k \leq r$
$is\ such\ that\ A[k] \geq pivot. Also\ we\ know\ that\ for\ all\ i, A[l] for\ p \leq l \leq i. A[l] \leq pivot$
$if\ i < j\ then\ there\ is\ no\ risk\ of\ acessing\ any\ element\ Now\ lets\ L\ be\ the\ time\ in\ which$
$j\ becomes\ equal\ i, in\ this\ situation\ we\ know\ that\ A[j] will\ be\ lower\ than\ the\ pivot, so\ it\ stops$
$The\ same\ happens\ for\ i\ becoming\ equals\ j$

d) When HOARE − PARTITION terminates, it returns a value j such that $p \leq j < r$

$Since\ we\ will\ not\ access\ any\ element\ outside\ of\ the\ array\ and\ just\ access\ the\ variables$
$before\ returning\ the\ value\ j\ we\ can\ assure\ that\ p \leq j \leq r$
$Now\ lets\ verify\ why\ cannot\ be\ j = r. j = r\ only\ in\ the\ firt\ iteraction\ so\ A[r] must\ be$
$\leq Pivot. But\ the\ in\ the\ first\ iteraction\ A[i] is\ always\ equal\ A[p] so\ it\ always\ equal\ pivot$
$We\ then\ will\ realize\ a\ swap\ between\ r\ and\ p\ and\ since\ r = p\ only\ if\ there\ is\ only$
$one\ element\ this\ doesn't\ occur.$

e) Every element of $A[p:j]$ is less than or equal to every element of $A[j+1:r]$ when HOARE − PARTITION terminates

$For\ every\ A[j] that\ we\ accessed\ A[j] \leq pivot\ and\ we\ swapped\ it\ with\ an\ element\ A[i]$
$such\ that\ A[i] \geq pivot. So\ given\ an\ j\ all\ elements\ above\ it\ are\ greater\ than\ the\ pivot$
$and\ every\ element\ simillarly\ will\ be\ lower. Now\ we\ will\ prove\ it\ by\ proving\ the\ following$
$loop\ invariant$:
$Prior\ to\ every\ iteration\ we\ have\ that\ A[p:i] is\ lower\ or\ equal\ than\ the\ pivot\ and$
$A[j:r] is\ always\ greater\ or\ equal\ than\ the\ pivot$

Initialization
$The\ elements\ are\ non\ existing\ so\ they\ trivially\ assure\ the\ assumption$
Maintenenace

Suppose it works for the following $p-1$ iterations of the while loop,
lets prove it's valid for the $p-th$ one. So in the beginning of the $p-1$ iteration
we say we have $i=o$ and $j=u$. we know that $A[p:o] \leq pivot \leq A[u:r]$
Now $j=j-L$ where L is some constant $\in$ Integers such $A[j-p] > pivot$ for $1 \leq p < L$ and
$j-L$ is such that $A[j-L]$ is $\leq x$. So our j value is now $o-L$.
$i=i+V$, where V is some constant $\in$ Integers such $A[i+Z] < pivot$ for $1 \leq Z < V$ and $A[i+V] \geq x$. Now we swap this elements, so $A[j-L] \geq x$ and $A[i+V] \leq x$
In the $p-th$ iteration we now have that $A[o:r]$were such that $A[o:r] \geq x$
We virified that now $A[o-p:r]$for $1 \leq p \leq L$ is greater than the pivot
So $A[o-L:r]$is greater than the pivot, but in this iteration $j=o-L$ so
$A[j:r]$ is greater than the pivot. The same argument goes to i
In the termination step the same holds

The PARTITION procedure in Section 7.1 separates the pivot value(originally in $A[r]$)
from the two partition it forms. The HOARE − PARTITION procedure,
on the other hand, always places the pivot value(originally in $A[p]$)
into one of the two partitions $A[p:j]$and $A[j+1:r]$since $p \leq j < r$,
neither partition is empty

f) Rewrite the QUICKSORT procedure to use HOARE − PARTITION

HOARE − PARTITION just give us the position such that , until there, all elements are lower o equal the pivot and after it all elements are greater or equal the pivot, but the pivot is not necessarily there so we must to put it there before recurse

---

7.2 *Quicksort with equal elements values*
The analysis of the expected running time of randomized quicksort in Section 7.4.2 assumes that all element values are distinct. This problem examines what happens when they are note
a)Suppose that all element values are equal. What is randomized quicksort's running time in this case

If all are equal we end up with the recurrence $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

b)The PARTITIONING procedure returns an index q such that each element of $A[p:q-1]$is less than or equal to $A[q]$and each element of $A[q+1;r]$ is greater than $A[a]$. Modify the PARTITIONING procedure to produce a procedure $PARTITIONING'(A,p,r)$, which permutes the elements of $A[p:r]$and returns two indices q and t, where $p \leq q \leq t \leq r$ such that

all elements of $A[q:t]$are equal,
each element of $A[p:q-1]$is less than $A[q]$ and
each element of $A[t+1:r]$is greater than $A[q]$

Like Partition, your Partition'procedure should take $\Theta(r-p)$time
We have the actual PARTITIONING procedure as below:

```
PARTITION(A, p, r)
1  x = A[r]                          // the pivot
2  i = p − 1                         // highest index into the low side
3  for j = p to r − 1                // process each element other than the pivot
4      if A[j] ≤ x                   // does this element belong on the low side?
5          i = i + 1                 // index of a new slot in the low side
6          exchange A[i] with A[j]   // put this element there
7  exchange A[i + 1] with A[r]       // pivot goes just to the right of the low side
8  return i + 1                      // new index of the pivot
```

$PARTITION'^{(A,p,r)}$:
$x = A[r]$
$i = p - 1$
$e = p - 1$
$for\ j = p\ to\ r - 1$
  $if A[j] < x$:
    $a = A[j]$
    $A[j] = A[e + 1]$
    $A[e + 1] = A[i]$
    $A[i] = a$
    $i = i + 1$
    $e = e + 1$
  $if\ A[j] == x$:
    $e = e + 1$
    $exchange\ A[e] with\ A[j]$
$exchange\ A[e + 1] with\ A[j]$
$return\ i + 1$

This have equal time as our previous PARTITION procedure, except for a constant factor coming from the new operations

c) Modify the $RANDOMIZED - PARTITION$ procedure to call $PARTITION'$, and name the new procedure $RANDOMIZED - PARTITION'$. Then modify the $QUICKSORT$ procedure to produce a procedure $QUICKSORT0\ .A;\ p;\ r$ that calls $RANDOMIZED - PARTITION0$ and recurses only on partitions where elements are not known to be equal to each other

$QUICKSORT'(A, p, r)$
$if\ p < r$
  $(q, t) = RANDOMIZED - PARTITION'$
  $QUICKSORT(A, p, q - 1)$
  $QUICKSORT(A, t + 1, r)$

d) Using $QUICKSORT'$ adjust the analysis in section 7.4.2 to avoid the assumption that all elements are distinct

$PARTITION'$ have the same running time and the subproblems are smallers then previous so our new quicksort method are in the worst case as worse as the expected running time

7.3 *Alternative quicksort analysis*

An alternatiev analysis of the running time of randomized quicksort focuses on the expected running time of each individual recursive call to RANDOMIZED
− QUICKSORT, rather than on the number of comparisons performed
As in the analysis of Section 7.4.2, assume that the values of the elements are distinct.

a) Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $\frac{1}{n}$. Use te this probability to define indicator random variable $X_i = I\{i\ th\ smallest\ element\ is\ chosen\ as\ the\ pivot\}$. What is $E[X_i?]$

well, given any element, $RANDOMIZED - PARTITION$ has a $\frac{1}{n}$ probability of placing into the pivot position $A[r]$. Now Let $X_i =$

$X_i = I\{i\ th\ smallest\ element\ is\ chosen\ as\ the\ pivot\}$. In this situationits clearly $\frac{1}{n}$

b) Let $T(n)$ be a random variable denoting the running time of quicksort on an array of size n. Argue that:

$$\boldsymbol{E[T(n)] = E[\sum_{q=1}^{n} X_q\, T((q-1) + T(n-q) + \Theta(n))]}$$

Well we know that $X_q = 1$ only if q was chosen as the pivot, obviously all the others will be 0, so $T(n)$ clearly is $\sum_{q=1}^{n} X_q\, T((q-1) + T(n-q) + \Theta(n))]$ for a given q chosen as pivot. We them can take expectation on both sides

c) Show how to rewrite equation above as $E[T(n)] = \frac{2}{n}\sum_{q=1}^{n-1} E[T(q)] + \Theta(n)$

By the linearity of expectation and independence we have:

$$E[T(n)] = \sum_{q=1}^{n} E[X_q\, T((q-1) + T(n-q) + \Theta(n))]$$

$$= \sum_{q=1}^{n} E[X_q] E[\, T((q-1) + T(n-q) + \Theta(n))]$$

$$= \sum_{q=1}^{n} \frac{1}{n} E[T((q-1) + T(n-q) + \Theta(n))] =$$

$$= \frac{1}{n} E \sum_{q=1}^{n} T(q-1) + \frac{1}{n} E \sum_{q=1}^{n} T(n-q) + \Theta(n) \rightarrow we\ know\ that$$

$$\sum_{q=1}^{n} T(n-q) = \sum_{q=1}^{n} T(q-1) \rightarrow \frac{2}{n} E \sum_{q=1}^{n} T(q-1) + \Theta(n) = \frac{2}{n}\sum_{q=0}^{n-1} E[T(q)] + \Theta(n)$$

d*)*Show that $\sum_{q=1}^{n-1} qlg(q) \leq \frac{n^2}{2} lg(n) - \frac{n^2}{8}$ ***for*** $n \geq 2$

We can divide this summation in other two:

$$\sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} qlg(q) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} qlg(q) \leq lg\left(\frac{n}{2}\right) \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + lg(n) \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \rightarrow$$

$$= \lg(n) \sum_{q=1}^{n-1} q - \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \leq \frac{n(n-1)}{2} \lg(n) - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} \leq \frac{n^2}{2} \lg(n) - \frac{n^2}{8}, if\ n \geq 2$$

e*)*Using the bound from equation $\mathbf{7.4}$***, show that the recurrence in equation of the letter c has the solution*** $\boldsymbol{E[T(n)]}$
$= \boldsymbol{O(nlgn)}.$ ***(Hint: show, by substitution, that*** $\boldsymbol{E[T(n)]}$
$\leq \boldsymbol{anlg(n)})$ ***for sufficient large n and for some positive constant a***

$$\frac{2}{n} \sum_{q=1}^{n-1} E[T(q)] + \Theta(n)$$

We will assume that $E[T(n)]$*will be less that* $anlg(n) + b$
Now lets make a lot so substitution

$E[T(n)] = \frac{2}{n} \sum_{q=1}^{n-1} E[T(q)] + \Theta(n) \leq \frac{2}{n} \sum_{q=1}^{n-1} (aqlgq + b) + \Theta(n)$ *this because for the strong induction hypothesis we supposed that for all the values of* $T(k)$*such* k
$\leq n - 1$ *the hypothesis will hold and based on that we will prve that it holds for* n

$$= \frac{2a}{n} \sum_{q=1}^{n-1} qlg(q) + \frac{2ab(n-1)}{n} + \Theta(n)$$

$$\leq \frac{2a}{n}\left[\frac{n^2}{2} - \frac{n^2}{8}\right] + \frac{2b(n-1)}{n} + \Theta(n) < anlg(n) + b + \left(\Theta(n) + b - \frac{an}{4}\right) \leq anlgn + b$$

To the inequality above holds we need that $\left(\Theta(n) + b - \frac{an}{4}\right)$ *be less than* 0
This is clearly attainable. Based on these arguments, we choosing properly n_0, c *and* c_1 *for the* $\Theta(n)$*function we are done*

7.4 ***Stooge sort***
Professors Howard, Fine and Howard have proposed a deceptively simple sorting alrothm, named sooge sort in ther honor, appearing on the following page.

a*) *Argue that the call $\boldsymbol{StoogeSort(A, 1, n)}$ ***correctly sorts the array*** $\boldsymbol{A[1:n]}$

b*) *Give a recurrence for the worst − case running time of $\boldsymbol{STOOGE - SORT}$ ***and a tight asymptotic*** $(\boldsymbol{\Theta - notation})$ ***bound on the worst case running time***

c*) *Comapre the worst − case running time of $\boldsymbol{STOOGE - SORT}$ ***with that of insertion sort, merge sort, heapsort and quicksort. Do the professor deserve tenure***?

```
STOOGE-SORT(A, p, r)
1  if A[p] > A[r]
2      exchange A[p] with A[r]
3  if p + 1 < r
4      k = ⌊(r − p + 1)/3⌋          // round down
5      STOOGE-SORT(A, p, r − k)     // first two-thirds
6      STOOGE-SORT(A, p + k, r)     // last two-thirds
7      STOOGE-SORT(A, p, r − k)     // first two-thirds again
```

$a)$ *Lets verify the correctness by first verifying if makes sense*
p *must be less than* r
We will call recursively if $p+1-p+1<r-p+1\rightarrow$
$2<r-p+1\rightarrow$ *this means that* $3\leq r-p+1$*(since they are integers)*
$1\leq\frac{r-p+1}{3}\rightarrow$ *this is the* min *and* max *possible values for* k*, in this situation*:
$p, r-k\rightarrow p, \frac{2r+p-1}{3}$ *is* $p>$*? suppose it is, then* $3p>2r+p-1\rightarrow 2p>2r-1$
this would mean that $p>r-\frac{1}{2}\rightarrow$ *meaning that* $p+\frac{1}{2}$
$>r$*, but we already verified that* $p+1<r$*, so its a contradiction*
is $p+k$ *greater than* r*? is so we would have* $2p+r+1>3r\rightarrow p+\frac{1}{2}$
$>r$*, what again is not the case*

We then verified that p *is always less than* r *in the recursive call*

$r-p+1$
$=n$*, where* n *is the size of the array(we just rearrange the indices, if necessary, to match* $1\ldots n$*)*
so $k=\left\lfloor\frac{n}{3}\right\rfloor$*. After the first recursive call they are either at the rightmost* $\frac{n}{3}$
or at the $\frac{n}{3}$ *rightmost of the* $\frac{2}{3}$ *leftmost. After that we call the sort in the last two thirds*
What guarantees that the largest are now at the rightmost of the whole array
After tht we realize another call to the first two thirds to adjust Given this intuition we can prove it rigorously in the following:

Lets prove it by induction, if $p=r$ *so the size* n *of the array is* $=1$
Then stooge sort does nothing and the array is already sorted.
For the case where $n=2$ *we have the following.* $[a,b]$
Suppose that $a<b$*. In this case* $p+1=r$ *and we doesn't enter in the if's condition statements and the array is already sorted. if* $a=b$ *the same happens.*
if $a>b$ *we swap then by the first line if condition, so we endup with a sorted array.*
Now suppose it works for $1\leq k<n-1$ *lets prove it sorts for* n
first situation of this call $A[p]<A[r]$
we also assume that $n>2.k=\left\lfloor\frac{n}{3}\right\rfloor$ *we then will call* $A(p,r-k)$
since $r-p+1=n\rightarrow r-k-p+1<n$ *so it will sort the array*
The same goes for the call of $p+k, r$ *and the last call*
We end up after the first recursive call with the array $A[p{:}\,r-k]$ *sorted*
This means that $A[p{:}\,p+k-1]$*is lower than* $A[p+k{:}\,r-k]$
We then do the second call making $A[p{:}\,p+k-1]$ *unchanged,* $A[p+k{:}\,r-k]\leq A[r-k+1{:}\,r]$*. we have in this subarray* k *elements that are higher than*

all elements in subarray $[p: p+k-1]$. *This means that after realizing the second recursive call the last k elements on the array* $A[p:r]$ *are* 100% *to have elements greater than all elements of the subarray* $A[p: r-k]$
so the last sort will make everything sorted

b) The recurrence is obviously $T(n) = 3T\left(\frac{2n}{3}\right) + \Theta(1)$.
Using the master theorem we arrive at a time complexity of $T(n) = \Theta\left(n^{\log_{\frac{3}{2}} 3}\right)$

c) this is asymptoptically greater than all worst cases already seen. no tenure for them

---

7.5 ***Stack depth for quicksort***
The QUICKSORT procedure of section 7.1 makes two recursive calls to itself. After QUICKSORT calls PARTITION, it recursively sorts the low side of the partition and then it recursively sorts the high side of the partition. The second recursive call in QUICKSROT is not really necessary, because the procedure can instead use an iterative control structure. This transformation technique, called tail − recursion eleminatio, is provided automaticaly by good compilers. Applying tail recursion elminiation transforms QUICKSORT into the TRE − QUICKSORT

```
TRE-QUICKSORT(A, p, r)
1  while p < r
2      // Partition and then sort the low side.
3      q = PARTITION(A, p, r)
4      TRE-QUICKSORT(A, p, q − 1)
5      p = q + 1
```

a) Argue that TRE − Quicksort $(A, 1, n)$ ***correctly sorts the array*** $A[1:n]$
Compilers usually execute recursive procedures by using a stack that contains pertinent information, including the parameter values, for each recursive call The information for the most recent call is at the top of the stack and the information for the initial call is at the bottom. When a procedure is called, its information is pushed onto the stack, and when it terminates, its information is popped. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stackrequires $O(1)$ ***stack space.***
The stack depth is the maximun amout of stack space used at any time during a computation.

b) describe a scenario in which TER − QUICKSORT's stack depth is $\Theta(n)$ ***on an*** n ***− element input array***

c) Modify TRE − QUICKSORT so that the worst case stack depth is $\Theta(\lg(n))$

a) We can prove it by induction, if $n = 1$ *we doesn't even enter the while loop so it works, for* $n = 2, p = 1$ *and* $r = 2$. *if* $A[2] > A[1]$ *then q will be* 2 *and we will call the recursion with* $A(1,1)$ *it will run nothing, it will make p equals* 3 *and the while loop won't iterate again. If* $A[1] > A[2]$ *then* $q = 1$ *and no other call will occur. In partition we swap these elements and they are now in order.*

Lets now assume that it works for $1 \leq k < n$. Lets look at our recursion now if q be any position we will recurse on $TREE - QuickSort$ of size $k < n$ and they all will correctly sort the arrays

b) This may occur when q is always r, in this situation our recurrence turns to be $T(n) = T(n-1) + \Theta(n)$. We then realize n recursive calls. Since we are calling the classic partition method this means that an array alrady sorted in increasing order will generate this result

c)There is no way that this is possible, since the partitioning can always get the last element In this situation and the depth will be n again

---

7.6 *Media − of − 3 partition*

***One way to improve the RANDOMIZED − QUICKSORT procedure is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. A common approach is meadian − of − 3 method*:**
choose the pivot as the meadin of a set of 3 elements randomly selected from the subarray. For this problem assume that the n elements in the input subarray $A[p:r]$ are distinct and thatn ≥ 3. Denote the sorted version of $A[p:r]$ by $z_1, z_2 \dots z_n$. Using the meadian − of − 3 method to choose the pivot element x, define $p_i = Pr\{x = x_i\}$

a) Give an exact formula for p_i as a function of n and i for $i = 2, 3 \dots, n-1$ Observe that $p_1 = p_n = 0$

***b) By what amount does the median − of − 3 method increase the likelihood of choosing the pivot to be $x = z_{\lfloor \frac{n+1}{2} \rfloor}$, the median of $A[p:r]$, compared with the ordinary implementation*?**
Assume that $n \to \infty$, and give the limiting ratio of these probabilities

c)Suppose that we define a* good *split to mean choosing the pivot as $x = z_i$ where $\frac{n}{3} \leq i \leq \frac{2n}{3}$.
***By what ammount does the median − of − 3 method increase the likelihood of getting a good split compared with the ordinary implementation*?**
(*Hint*: *Approximate the sum by an integral*)

d)Argue that in the $\Omega(nlgn)$ running time of quicksort, the median − of −3 method affects only the constant factor

a) for a given element p_i we will have $i-1$ elements to its left and $n-i$ elements to its right

The number of different trios we can form is $n \cdot (n-1) \cdot \frac{n-2}{3!}$.
Now for a given i, the number of different trios we can form such that i is choosen as the median is equal to $(i-1) \cdot (n-i)$. What leads us to

$$\Pr\{x = z_i\} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$$

b) In the random first method implemented the likelihood is $\frac{1}{n}$. Now we will suppose that

n *is odd, in this situation the* $\left\lfloor \frac{n+1}{2} \right\rfloor = \frac{n+1}{2}$ *so, the probability of* $x = z_{\frac{n+1}{2}}$ *is equal*:

$$\frac{6\left(\left(\frac{n+1}{2}\right)-1\right)\left(n-\left(\frac{n+1}{2}\right)\right)}{n(n-1)(n-2)} = 6\left(\frac{n-1}{2}\right)\left(\frac{n-1}{2}\right) = \frac{3(n-1)^2}{2n(n-1)(n-2)} = \frac{3(n-1)}{2n(n-2)}$$

So lets get the differences between the probabilitities:

$$\frac{3(n-1)}{2n(n-2)} - \frac{1}{n} \to \frac{3n-3-2n+4}{2n(n-2)} = \frac{n+1}{2n(n-2)}$$

Now the limit ration between these two probabilities is:

$$\lim_{n\to\infty} \frac{3}{2}\frac{n-1}{n-2} = \frac{3}{2}$$

Now lets verify for n being even, so we want the value $z_{\frac{n}{2}}$ *in this situation we have the following*:

$\frac{6\left(\frac{n-2}{2}\right)\left(\frac{n}{2}\right)}{n(n-1)(n-2)} = \frac{6(n-2)n}{4n(n-1)(n-2)} = \frac{3}{2(n-1)}$, *the limit again is clearly* $\frac{3}{2}$ *and the difference is* $\frac{3n-2n+2}{2(n-1)n} = \frac{n+2}{2n(n-1)}$

c) So a good split can be anyone between the range specified, with this the probability that one of them occurs is their sum: $\sum_{i=\frac{n}{3}}^{\frac{2n}{3}} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} \to$

$$\frac{6}{n(n-1)(n-2)}\sum_{i=\frac{n}{3}}^{\frac{2n}{3}}(i-1)(n-i) = \frac{6}{n(n-1)(n-2)}\sum_{i=\frac{n}{3}}^{\frac{2n}{3}} in - i^2 - n + i$$

Since we were given the hint to approximate the sum by an integral we can discard *the cases where n is not a multiple of* 3. *We have* $\frac{6}{n(n-1)(n-2)}\int_{\frac{n}{3}}^{\frac{2n}{3}} -i^2 + (n+1)i - n$

$$= \frac{6\left[-\frac{7n^3}{81} + \frac{n^3}{6} + \frac{3n^2-6n^2}{18}\right]}{n(n-1)(n-2)} = \frac{\frac{6}{81}[-7n^3 + 13.5n^3 - 13.5n^2]}{n(n-1)(n-2)} = \frac{39n^3 - 81n^2}{81n(n-1)(n-2)} \to$$

$\frac{13n^2-27n}{27(n-1)(n-2)}$. *if n goes to infinity our probabiliy becomes* $\frac{13}{27}$ *which is larger than* $\frac{1}{3}$

Which is the ordinary implementation

d) We still use the partition method, but just make a better approach, the time is still the same in terms of complexity, so our recurrence is still the same, doing $O(n)$*work in each level, which leads us to the same case*

7.7 *Fuzy soring of intervals*

Consider a sorting problem in which you do not know the numbers exactly. Instead, for each number, you know an interval on the real line to which it belongs. That is, you are given n closed intervals of the form $[a_i, b_i]$, ***where*** $a_i \le b_i$. ***The goal is to fuzzy − sort these intervals: to produce a permutation*** $< i_1, i_2 \dots i_n >$ ***of the intervals such that for*** $j = 1, 2 \dots .$, ***n ther eexist*** $c_j \in [a_{ij}, b_{ij}]$ ***satisfying*** $c_1 \le c_2 \le c_3 \dots \le c_n$

a) Design a randomized algorithm for fuzzy sorting n intervals. Your algorithm should have the general structure of an algorithm that quicksorts the left endpoinds (the a_i values), but it should take advantage of overlapping ntervals to improve the running time. (As the intervals overlap more and more, the problem of fuzzy sorting) the intervals becomes progressively easier. Your algorithm should take advantage of such overlapping to the extent that it exists

b) Argue that your algorithm runs in $\Theta(nlgn)$ expected time in general, but runs in $\Theta(n)$ expectedtime when all of the intervals overlap (ie.., when there exists a value x, such that $x \in [a_i, b_i]$ forall i). Your algorithm should not be checking for this case explicitly, but rather its performance should naturally improve the amount of overlap increases

a) and b) We can perform the following algorithm:
$FUZZY - PARTITIONING(A, p, r)$:
$x = A[r]$
$i = p - 1$
$h = p - 1$
$for\ j = p\ to\ r - 1$:
  $if\ b_j < x.a$:
    $i = i + 1$
    $h = h + 1$
    $y = A[j]$:
    $A[j] = A[h]$
    $A[h] = A[i]$
    $A[i] = y$
  $if\ b_j \geq x.a\ and\ a_j \leq x.b$:
    $x.a = \max\{x.a, a_j\}, x.b = \min\{x.b, b_j\}$
    $h = h + 1$
    $swap\ A[h], A[j]$
$i = i + 1$
$swap\ A[i] with\ A[r]$
$h = h + 1$
$return\ (i, h)$
Ok, so the idea of this algorithm is basically the same produced in the exercise 7.2
We will treat the overlapping intervals equal because it doesn't matter who comes before or after since we can choose some element from both that satisfies the fuzzy sort output
Besides it when we find overlapping intervals we change our $x.a$ and $x.b$ so that our interval is such that any intervaçl that overlaps with it will also overlap with the other 2 and we can consider all of them equal

The expected running time is suppose to be the same as the one we have taken the idea
Now let's write the FuzzySort because we just have choosen the partition

$FuzzySort(A, p, r)$:
$if\ p < r$:
  $a, b = FUZZY - PARTITIONING(A, p, r)$
  $FuzzySort(A, p, a - 1)$
  $FuzzySort(A, b + 1, r)$

Now since all elements are overlaping we will always fall on the second if statement
What leads us to end with $i = p$ and $h = r$.

$We\ will\ then\ call\ FuzzySort(A, p, p-1) and\ FuzzySort(A, r+1, r)$
$Which\ will\ not\ recurse\ anymore, leading\ to\ a\ \Theta(n)\ complexity$