HW3

1. A.)

minMax(A)

j=0

initialize minarray and maxarray

for i=0 to A.length

x=A[i]

y=A[(i+1)%n]

i=i+2

if(min<max)

minarray[j]=x

maxarray[j]=y

j++

else

minarray[j]=y

maxarray[j]=x

j++

min=minarray[0]

max=maxarray[0]

for i=1 to minarray.length

if minarray[i]<min

min=minarray[i];

i++

for i=1 to maxarray.length

if minarray[i]>max

max=maxarray[i]

i++

B.) In the beginning of the algorithm, comparing the values to separate the array into minarray and maxarray is n/2 comparisons. Next, comparing the values within the array minarray takes n/2 comparisons. Finally, comparing in the max array takes n/2 comparisons.

Therefore, n/2+n/2+n/2=3n/2

2.)

a.) Dividing into 7 would have 4 elements greater than x. Therefore leading to

Looking at step 5 has a size

The Recurrence would then be T(n) assume T(n)

n>83 we can find a constand that satisfy this equation

let n=126

14a

Then find a constant a and c such that no=140 thue T(n)=O(n)

b) Select in groups of 3

Elements greater than x

Step 5

Recurrence

c>0

1. A.)

QUICKSORT(A,p,r)

1. If (p<r)
2. n=h-p+1 //size of subarray
3. m=med(A,p,r,n/2) //median of array
4. q=partition(A,p,r,med)
5. QUICKSORT(A,p,q-1)
6. QUICKSORT(A,q+1,r)

b.)T(n)=2T(n/2)+O(n)

c) Calling the partition in quicksort takes in the median of the input array. This is the pivot element. The worst-case of SELECT is O(n). Partition is now split into two which guarantees best case partitioning. Thus giving us the recurrence shown at (b)