Text

Description automatically generated

Based on the insertion sort algorithm, the run time complexity of sorting the arrays would be O(nk),

If we use quick sort untilt he problem size becomes smaller than k, we need lg(n/k) step,

In theory, we should pick k to minimize the complexiy, where k could satisfy the equation:

C as constant

c\_1 \* nlgb >= c\_2 n k + c\_1 n lg(n/k)

So that we have lgk >= c2/c1 \* k

In practice, we need a huge data set to try to find the best various values of K.

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A,

Since all elements are equal, randomized quicksort, the run time would always q = r so that the time would be T(n) = T(n- 1) + O(n) = O(n ^ 2)

*B,*

PARTITION'(A, p, r)

x = A[p]

low = p

high = p

for j = p + 1 to r

if A[j] < x:

y = A[j]

A[j] = A[high + 1]

A[high + 1] = A[low]

A[low] = y

low = low + 1

high = high + 1

else if A[j] == x

exchange A[high + 1] with A[j]

high = high + 1

return (low, high)

C,

QUICKSORT'(A, p, r)

if p < r

(low, high) = RANDOMIZED-PARTITION'(A, p, r)

QUICKSORT'(A, p, low - 1)

QUICKSORT'(A, high + 1, r)

D,

**3 similar to figure 8.2, illustrate the operation of counting-sort on**

**A = [5,7,3,1,3,6,2,1,3,5]**

**Answers:**

First we need to create a count array to store the count of each unique object,

Then we modigy the count array such that each element at each index stores the sum of previous counts,

The modified count array indicateds the position of each object in the outpout sequences,

Then we rotates the array clockwise for one time,

The, Output each object from the input sequence followed by

increasing its count by 1

the process would be that :

count array:

[0, 0, 0, 0, 0, 0, 0]

[2, 1, 3, 0, 2, 1, 1]

[2, 3, 6, 6, 8, 9, 10]

[0, 2, 3, 6, 6, 8, 9]

Outputarray =

[0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[1, 1, 2, 3, 3, 3, 5, 5, 6, 7]

Graphical user interface, text

Description automatically generated

Any number in the n numbers could be either the max one or the min one at the original status.

Set max array and min array,

We pick the items from the list of numerbers, a, and b, if a <= b, we remove a from max and remove b from min, reduce the counts of both sets by 1, there are n/2 comparisions until min and max are disjoint, and the sets will have size of n/2, we also could compare in the max array and in the min array and it will only reduce either one array by one.

In total there are three types of comparison,

Therefore, we require a total of n/2 – 1 + n/2 – 1 = n – 2 comparison, adding the first one, n – 2 + n/2 = 3/2n – 2 in the worst case.

Graphical user interface, text, application, letter, email

Description automatically generated

A,

The algorithm would first sort the numbers, the best asymptotic worst case, would be n\*log(n), and the listing the I largest would take O(i) with a for loop, therefore the total running time would be. O (nlog(n) + i)

B, since we are building the max-priority queue, we are calling healpify process, this will take nlog(n) in the worst case in the best way, the call of extract-max with I times, would be I \* log(n)

Therefore the total time would be O(nlog(n) + I \* log(n))

C, the partitioning around the I th largest number would take O(n), sorting the subarray of lengthi would be ilog(i) so the total time would be O(n + ilog(i))