20407 Data Structures and Introduction to Algorithms

Maman 11

Robert S. Barnes

Question 1

1. Assume n is even. The given list, is made of 2 ordered lists, the first containing numbers, and the second containing numbers. Thus there are inversions in the list.

Compares: On the first iterations, only one compare is made since the elements are already in order. On the iteration we encounter 1. The inner loop executes compares as the first element are shifted up one place. Then the loop is stopped by the condition. At the end of the outer loop, inversions have been removed. On each subsequent outer loop, the inner loop executes comparisons, since it is now stopped by the A[i]>key condition. Thus we have:

The equation doesn't hold for since the array is already sorted for that value.

The number of key compares is and we see that the following hold:

Copies: The algorithm executes 2 copies for each iteration of the outter loop, regardless of order which gives us copies. Then the inner loop effectively removes one inversion per copy per iteration thus:

The number of key copies is and we see that the following hold:

1. Assume n is even. We see that for each value of , there are inversions in the list.

Compares: On each iteration of the outer loop, inversions are removed resulting in the same number of compares, plus one additional compare to stop the loop. So we have:

The number of compares in and we see that the following holds:

Copies: The routine performs one copy per inversion removed to shift elements up, plus 2 copies for the key on every loop regardless and thus we get:

The number of copies is and we see that the following holds:

Question 2

1. We'll show that if , then:

Assume that for we have and that all are asymptoticlly positive, since otherwise the sets are empty and the claim is trivially true. This implies that:

For simplicty assume . We look for such that:

By assumption, all the are asymptotically positive, so we can divide by :

The first part of the inequality holds since each is asymptotically positive. The second part holds because which implies that for each term in the summation holds. In fact, since equals some at each point we have that:

and thus:

by transitivity.

1. Assume:
2. Assume

Question 3

1. By definition, r(A[i]) equals the total number of elements in A which are less than A[i], plus the number of elements to the left of A[i] which are equal to A[i], plus one. But this is nothing more than the index of A[i] in A after performing a stable sort. Since the indexes of each element in A after performing a stable sort are simply , then R must be a permutation of these numbers.
2. For an array of length 1, Rank(A,R) produces the correct output since A[1] <= A[1] and we get R[1]=1.

Assume that for the two loops produce the correct output. On the next outter loop and the inner loop advance through A comparing each element to A[n]. Before each iteration of the inner loop starts, R[i] contains the rank of A[i] in the array A[1..n-1] for . For , if A[i]>A[n], then R[i] is incremented and has it's final correct value. Otherwise, if A[i]<=A[n], then R[n] is incremented. This happens once for each value in A[1..n-1] which is less than or equal to A[n] and then once more when j=n since A[n]<=A[n]. Thus, at the end of the loop R[n] contains the rank of A[n], and R[1..n-1] contain the corresponding ranks of A[1..n-1].

1. R[i] contains the position of A[i] in A after performing a stable sort. Thus U[R[i]] = A[i] copies A[i] to it's correct position in A and then the second loop copies the sorted array U back into A.
2. In the call to RANK, the loops always execute

compares in all cases. In RANK-SORT, all the elements of A are copied to U, and then back to A, resulting in copies in all cases.

1. Since each R[i] contains the corresponding index of A[i] after performing a stable sort, then A is sorted if and only if R is sorted, since RANK-SORT1 performs identical swaps on the two arrays.

For an array of length one we have R[1]=1, i=1 and the function does nothing, R and A are sorted.

Assume that R[1..k] is sorted for some . Then R[k+1..n] contains some permutation of the numbers . If R[i]=k+1 then we're done and now R[1..k+1] is sorted. If not, then the value at position i is swapped with the value at position R[i], which is the correct position of the value at i. Since each iteration of the loop places at lesat one value in R[k+1...n] into the correct position, then the loop terminates after a maximum of iterations. At that point the correct value will be in R[k+1] meaning that R[1..k+1] and A[1..k+1] are now sorted.

F)

Worst Case:

Compares: Since RANK-SORT1 calls RANK, then it always performs exactly compares on A. Copies: 3(n-1) Because the last swap always puts 2 elements in their correct place and each previous swap places one element in it's correct position.

Best Case:

Cmps = Copies = 0

G)

4 3 1 2

2 3 1 4

3 2 1 4

1 2 3 4

1. Since RANK always takes compares to complete, then both algorithms worst case run time is . The space complexity is in both cases since both algorithms allocate R to pass to RANK.
2. Since the RANK(A,R) function always runs in time, then the best, worst, and average case times for both sorting algorithms is .

Question 4

1. Show that:
2. Assume that and that . We look for some value of such that:

The existance of the limit implies that is asymptotically positive, thus we can divide by and thus we are looking for some such that

The existance of the limit implies that for any constant value of which we choose, no matter how close to , we can always choose a value of such that is smaller and the above inequality will hold.

1. Assume that . implies that is asymptotically positive and that we can divide by . Thus no matter how close to we choose , we can always choose a value for such that . This implies

B,C) We have that

Thus and is a lower order term, implying that the following holds: