**CMPT435 Algorithm Design and Analysis**

**Spring 2017**

**Final Exam Example**

1. Show step by step how the quicksort algorithm sorts the array 4, 6, 3, 7, 1. Indicate at each step what the partitioning element is.

4, 6, 3, 7, 1.

(4, 6) (3, 7, 1)

(4) (6) (3, 7, 1)

.

(4, 6) (3, 7, 1)

(4, 6) (3) (7, 1)

(4, 6) (3) (7) (1)

(4, 6) (3) (1, 7)

(4, 6) (1, 3, 7)

(1, 3, 4, 6, 7)

1. The mode of an array is defined to be the number that occurs most frequently in the set. The set {4,6,2,4,3,1} has a mode of 4.

Design an algorithm that takes an array as input, and returns the mode of the input array. Full credit (15 points) will be awarded for an algorithm that is O(n log n). Algorithms that are O(n2 ) or slower will be scored out of 10 points.

(i) describe the idea behind your algorithm in English (5 points);

(ii) provide pseudocode (5 points);

(iii) analyze its running time (5 points).

Answer:

1) Call Quicksort() on the input array.

2) For each element x in the input array

a) use Binary search to get index of the first occurrence of x in L. Let the index of the first occurrence be i.

b) Use Binary search to get index of the last occurrence of x in L. Let the index of the last occurrence be j.

c) x occurs (j – i + 1) times.

3) Compare elements and find the one occurs most frequently.

FindMode (L)

Quicksort(L)

mode=L[0]

max = 1;

for k in [0, n-1]

i = first(L, L[k])

j = last(L, L[k])

times = j – i +1

if(times > max)

max = times

mode = L[k]

end if

end for

End FindMode

first( L, x)

start= 0, end = L.length-1;

while(start<=end)

{

middle = (start + end) / 2;

if (L[middle] == x)

first = middle;

end = middle-1;

else if (L[middle] > x )

end = middle-1;

else

start = middle+1;

}

return first;

}

last( L, x)

{

while(start<=end)

{

middle = (start + end) / 2;

if (L[middle] == x)

last = middle;

start = middle+1;

else if (L[middle] > x )

end = middle-1;

else

start = middle+1;

}

return last;

}

Time Complexity:

Quicksort() O(nlogn)

There are n elements in L, and for each element, we call (modified) Binary Search twice, thus 2\*O(log n)\*n

As a result, O(nlogn) + 2O(nlogn) = O(nlogn).

1. Two unsorted arrays, A, B.

A and B together contain n integers.

Output:

Identify the elements that appear in both A and B.

If an element appears in A, AND also in B, it should appear in the output.

For example, if A and B are:

{98, 2, 3, 1, 0, 0, 0, 3, 97}

{1, 3, 4, 2, 1, 0, 10, 98}

The output should be {0, 1, 2, 3, 98}, as these elements appears in both A and B.

Design an algorithm to solve this problem. Full credit (15 points) will be awarded for an algorithm that is O(n log n). Algorithms that are O(n^2 ) or slower will be scored out of 10 points.

(i) describe the idea behind your algorithm in English (5 points);

(ii) provide pseudocode (5 points);

(iii) analyze its running time (5 points).

Answer:

First, call Quicksort() on A and B to sort A and B.

Then, to find elements that occur in both A and B, we would create two pointers, i1 and i2, and let them point to the first element of A and B, respectively. We’ll compare A[i1] with B[i2], and output A[i1] (or B[i2]) only when A[i1] = B[i2].

FindCommon(A, B)

Quicksort(A)

Quicksort(B)

i1 = 0, i2 = 0

while (i1<A.length && i2 <B.length)

if A[i1] == B[i2]

output(A[i1]);

i1++;

i2++;

else if A[i1] < B[i2]

i1++;

else

i2++;

end if

End while

Running time: O(nlogn) + O(n) = O(nlog n)

1. Given three positive integers B, n and M as inputs, design an algorithm to compute the value of (Bn mod M) in O(log n) time in the worst case.

Assume we can calculate B mod M in O(1) time.

Note that Bp+q mod M = ((Bp mod M) \* (Bq mod M)) mod M.

Full credit (15 points) will be awarded for an algorithm that is O(log n). Algorithms that are O(n) or slower will be scored out of 10 points.

(i) describe the idea behind your algorithm in English (5 points);

(ii) provide pseudocode (5 points);

(iii) analyze its running time (5 points).

Answer:

Divide and Conquer

Bn mod M = Bn/2 + n/2 mod M

= ((Bn/2 mod M) \* (Bn/2 mod M)) mod M

= (Bn/2 mod M)2 mod M

Similarly,

Bn/2 mod M = Bn/4 + n/4 mod M

= ((Bn/4 mod M) \* (Bn/4 mod M)) mod M

= (Bn/4 mod M)2 mod M

Base case:

n = 1

Bn mod M = Bmod M

General case:

n > 1

Bn mod M = Bn/2 + n/2 mod M

= ((Bn/2 mod M) \* (Bn/2 mod M)) mod M

= (Bn/2 mod M)2 mod M

ComputBM(n)

if n = 1 then

return B mod M;

else

return Math.pow(ComputeBM(n/2), 2);

endif

The time complexity of this algorithm is O(log n).

1. Using iteration, find a tight bound for the solution of the following recurrence equation.

T(n) = 2T(n/2) + n, n>1. Assume T(1) = 1, and (n is a power of 2) (5 points).

Answer:

See slides “Sorting 2” on iLearn, page 16 - 17

1. We are given an array of n numbers A in an arbitrary order. Design an algorithm to find the largest and second largest number in A using at most 3/2n -2 comparisons.

Full credit (15 points) will be awarded for an algorithm that is O(log n). Algorithms that are O(n) or slower will be scored out of 10 points.

(i) describe the idea behind your algorithm in English (5 points);

(ii) provide pseudocode (5 points);

(iii) analyze its running time (5 points).

Answer:

Divide and Conquer

Base case:

When the list size is 1: the max and 2nd max refer to the same element in the list

When the list size is 2: compare 2 elements: the larger element is the maximum element, and the smaller one is the 2nd max.

General case:

When the list size is greater than 2,

1. Split the array in half:

left, right

1. Find the maximum element and the 2nd maximum element of each half:

maxleft, max2left, maxright, max2left

1. Compare maxleft with maxright: the larger one is the maximum element of the entire array.

Pick the smaller of maxleft and maxright, compare it with the 2nd max element of the other half, the larger one is the 2nd maximum element of the entire array.

As the algorithm is quite similar to that of minmax(), discussed in slides Divide and Conquer 2, page 9-11, it makes the same number of comparisons: 2/3\*n -2 compares.

Max2ndMax(A, start, end, max, max2)

{

if(end == start)

{

max = max2 =A [start];

}

else if (end == start + 1)

{

if (A[start] > A[end])

{

max = A[start]

max2 = A[end]

}

else

{

max = A[end]

max2 = A[start]

}

}

else

{

middle = start + (end - start) / 2;

Max2ndMax (start, middle, maxLeft, max2Left);

Max2ndMax (middle + 1, end, maxRight, max2Right);

max = (maxLeft > maxRight) ? maxLeft : maxRight;

temp = (maxLeft < maxRight) ? maxLeft : maxRight;

if(temp == maxleft)

{

max2 = (maxLeft > max2Right) ? maxLeft : max2Right;

}

else

{

max2 = (maxRight > max2Left) ? maxRight : max2Left;

}

}

}