D0012E
Lab 1

Group 86
Henrik Eklund*
Alex Bergdahl†
Emil Magnusson‡

Teacher comments:

December 2, 2021

*henekl-0@student.ltu.se
†alxber-0@student.ltu.se
‡emimag-0@student.ltu.se

Contents

1 Introduction

Our group have received an assignment, which is to create multiple algorithms and explain them. The first assignment was to create 2 separate algorithms for the same problem, one using incremental and one using the divide and conquer method. Where both are going to calculate the 3 smallest elements in a list of a minimum 3 elements.

The other problem, is to find the largest adjencent sublist sums using divide and conquer algorithm.

2 Theory

2.1 Incremental

The incremental approach starts by creating a list where we are supposed to hold our smallest elements and one variable to hold which one is the biggest of the 3.

We then loop through our list of elements. During this loop, we check if our temporary created list contains less than 3 elements and if it does we add the current X_i to our list and check if it's the biggest of the 3 elements.

When the initial list is filled, we continue with checking if the biggest of our sublist is larger than the next X_i and if it's we compare it to the 2 other elements, and place it into the correct position.

We then make sure our sublist of the elements is sorted correctly and update it if it's not, then we return our triples of the 3 smallest elements in the list

2.1.1 Induction proof

Precondition:

1. List of Elements with a length equal or bigger than 3
2. Assume the list contains only whole numbers. (1, 2, -3)
3. Assume n $= 3 * 2^{(k-1)}$

Postcondition:

1. A tuple (x, y, z) where $(x \leq y \leq z)$
2. Values x, y, z are the smallest numbers in the list of Elements

Base case:
Our base case $k = 1$ which gives us $n = 3 * 2^k - 1$ consist of only 3 elements, and by definition a 3-element array is already the 3 smallest elements and after is sorted. Thus the base case holds.

Induction Step:

Assume that $inc(p)$ is true. That is, for any array of length where $n = 3 * 2^p$ is true.

To prove that $inc(p + 1)$ is true.

Let a list of length $p + 1$ be our inserted list. First our loop will run p times. Which will find the smallest 3 elements on the first p elements. Then it will check one more time, to find check the last and final element. If it is smaller than one of our 3 smallest elements. We replace the biggest of them and place the value in the correct location.

If not. We leave the loop and return our tuple

Our Result for $inc(p + 1)$ will be our tuple with the smallest elements in our inserted list, and that they are in the correct order. thus we proved the correctness of our algorithm

2.1.2 Comparisons

As our basecase $n = 3$. Our code will only execute eight comparisons

Though, When we have $n > 3$ We will have a few more comparisons.

First we will include our base case of 8. We then have an if that is true whenever our temp list is filled. Then that gives it a comparison time of $n - 3$. Within this if statement contains a loop that will run 3 times. Which contains another if statement. Which gives this area $(n - 3) * 3$

Lastly we sort our result to get our required $x < y < z$ which gives us another $+2$ comparisons. Which when combined give us this a comparison count of about $3 * 2 + 2 + (n - 3) * 3$

2.2 Divide and Conquer

Our divide and conquer algorithm works by simply dividing the input array in half recursively until we reach a base case when it only contains three elements. After this base case is reached we return our elements since in an array of size three the three smallest elements are the only elements. We then have two arrays each of size three since we are combining the array of size three from left and array of size three on the right and that makes a

combined array of size six. We use our incremental function to find the three smallest elements on this one and then return them.

2.2.1 Induction proof

Precondition:

1. List of Elements with a length equal or larger than 3
2. Assume the list contains only integer numbers. (1, 2, -3)
3. Assume n = $3 * 2^{(k-1)}$

Postcondition:

1. A tuple (x, y, z) where ($x \leq y \leq z$)
2. Values x, y, z are the smallest numbers in the list of elements

Base case:

Like with the incremental sort, our base case occurs when k = 0 for n = $3 * 2^{(k-1)} = 3$. We then split our elements in two, left and right. The length of these list are always three.

We then get each element back. By using our incremental algorithm to sort the three elements. As definied in incremental induction proof. An 3-element array is already the 3 smallest elements. Thus the base case holds.

Induction Step:

We again assume that $divide(p)$ is true. That is, if the length of any array is $n = 3 * 2^p$.

To prove that $divide(p + 1)$ is true.

Let b be a list of length $p + 1$. First our algorithm checks if $p = 3$. As we want to prove $p + 1$, we know that our first itiration is larger than 3, as is covered in p. We then split our list both left and right using the middle point as pivot point.

The returning elements will always be three. And by using our incremental algorithm to sort the the elements. The returning list will contain our sorted smallest value. Thus our algorithm correctness is validated.

2.2.2 Comparisons

As stated above. Our basecase is $n = 3$, and our code will then have nine comparisons. Though when our n value is larger than three, our comparisons increased drastictly.

First our base case of nine is included. and as we split it twice, we double it. But also do to it being divide and conqurer. Each split adds $n/2$

we then get a comparison equation to $2 * (n/2) + 2 * (3 * 2 + 2 + ((n - 3) * 3))$ Which when $n = 3$ Do give us the correct value 19

2.3 Max Sub array

Our Max Sub array algorithm works by using divide and conquer. Splitting our list in the middle over and over, until we get our basecase.

We then start comparing left and right. Checking and finding its max value. After that we check if the max value on the left side is larger than the total sum of the left side and right sides max value, which means if left max value is larger then we set that as our max left and vice versa. This is to find our left max value. We then do the same for the right side to find the right max value.

Then in some special cases, we will find our max sub list in both the left and righ side, let us say position 16 is the middle and the max value is 14, 15, 16, 17 and 18. Then we check the right max value and compare it to the lefts most right max value and the rights most left max value, and by doing the other side aswell. If that statement is correct we then know if the max value is correct because it will be larger than any max value on either left or right side.

Repeating these steps until we find our final max sum.