Assignment 4 solutions

Responsible TA: Claudi Lleyda Moltó

Task 1 - Center of mass

Consider n particles in 3-dimensional space with weights w_i and positions $\vec{p} = (x_i, y_i, z_i)$ for $i \in \{1, \ldots, n\}$. We can calculate their center of mass with

$$\vec{P} = \frac{\sum_{i=1}^{n} w_i \vec{p_i}}{\sum_{i=1}^{n} w_i},$$

where the numerator is first moment of mass, and the denominator is the total mass of the objects.

Devise a simple divide-and-conquer algorithm to calculate their center of mass in $\mathrm{O}(n \log(n))$. Justify your answer.

Solution. Note how the problem consists on adding two arrays of n numbers: one for the first moment of mass and another for the total mass of the objects. We can perform these additions independently as two invocations of the same divide-and-conquer algorithm.

An algorithm to add an array of numbers using divide-and-conquer is as follows

```
function AddArray(Array array)
    n ← Size(array)
    if n = 1 then
        return array[0]
    sum_left ← AddArray(array[0 : n/2])
    sum_right ← AddArray(array[n/2 : n])
    return sum_left + sum_right
```

Every invocation of AddArray performs a constant amount of work for the base case, and may call itself recursively twice with half the input each time.

We have seen multiple problems like this, and we identify that the running time $T(n)$ of this algorithm satisfies

$$T(n) \leq 2T(n/2) + cn$$

for some constant $c > 0$, which gives us a runtime of $\mathrm{O}(n \log(n))$.

We have to run this algorithm a total of 4 times; one for each of the 3 dimensions of the first moment of mass, and an extra invocation for the total mass of the objects. Lastly, we must perform the final division in the formula to get the desired result.

All together, these are insignificant operations, and this gives us a big-O running time of $O(n \log(n))$, as desired. $\Diamond$

Task 2 - Majority element

A majority element in an array of n elements is an element that shows up at more than $\lfloor n/2 \rfloor$ times in the array.

Devise a divide-and-conquer algorithm to find the majority element in an array of size n. What is its big-O running time? Justify your answer.

Solution. We propose the following algorithm

```
function FindMajorityElement(Array array)
    n ← Size(array)
    if n = 1 then
        return array[0]
    if n = 2 then
        if array[0] = array[1] then
            return array[0]
        else
            return NULL
    count ← 0
    array_1 ← array[0 : n/2]
    array_2 ← array[n/2 : n]
    result ← FindMajorityElement(array_1)
    if result ≠ NULL then
        for element in array do
            if element = result then
                count ← count + 1
    if count < n/2 then
        count ← 0
        result ← FindMajorityElement(array_2)
        if result ≠ NULL then
            for element in array do
                if element = result then
                    count ← count + 1
    if count < n/2 then
        return NULL
    else
        return element
```

This approach works thanks to the observation that if there is a majority element, then that same element must also be a majority element in one of the two partitions.

Note how, in the worst case scenario, each invocation of FindMajorityElement calls itself twice, and tests $2n$ cards outside these recursive calls. Therefore we identify that the running time $T(n)$ of the algorithm satisfies

$$T(n) \leq 2T(n/2) + 2n,$$

which as we have seen before gives us a big-O running time of $O(n \log(n))$. $\Diamond$