Algorithms and Complexity. Exercise session 2

Greedy + Divide and Conquer

MST greedy Describe a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. Namely, it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

Solution to MST greedy
Prim's algorithm for computing the minimum spanning tree on a weighted graph. □

Max and min Let $v[1..n]$ be a vector of n integers. Give a recursive algorithm that computes the largest and the smallest number in v. The algorithm uses at most $\lceil 3n/2 \rceil - 2$ comparisons between elements in v. Note that the number of elements in v may not be a power of 2.

Solution to Max and Min
When you only have two numbers it takes is a single comparison to both find the largest and the smallest number.

```
MinMax(v,i,j)=
  if i=j then return (v[i],v[i])
  else if i+1=j then
     if v[i]<v[j] then return (v[i],v[j])
     else return (v[j],v[i])
  else
     m:=Floor((j-i)/2)
     if Odd(m) then m:=m+1;
     (min1,max1):=MinMax(v,i,i+m-1);
     (min2,max2):=MinMax(v,i+m,j);
     min:=(if min1<min2 then min1 else min2);
     max:=(if max1>max2 then max1 else max2);
     return (min,max);
```

The computation tree will have $\lceil n/2 \rceil$ leaves and $\lceil n/2 \rceil - 1$ internal nodes. If n is even, all $n/2$ leaves will make a comparison. If n is odd, there is one leaf (the rightmost) that requires no comparison. Therefore the leaves require $\lfloor n/2 \rfloor$ comparisons. In each internal node we made two comparisons, ie a total of $2\lceil n/2 \rceil - 2$ comparisons. In total, we get $\lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2 = \lceil 3n/2 \rceil - 2$ comparisons. It can be proved that the problem can not be solved with fewer comparisons. □

Matrix multiplication Strassen's algorithm multiplies two $n \times n$-matrices in time $O(n^{2.808})$ by decomposition of 2×2-block matrices. It is faster than $O(n^3)$ because it computes seven multiplications instead of eight to form the product matrix. Another idea is to make the decomposition of 3×3-block matrices instead. A researcher at NADA tried a couple of years ago to find the minimum number of multiplications needed to multiply two 3×3-matrices. He managed to get almost 22 multiplications. If he had succeeded, what would have been the time complexity to the multiplication of two $n \times n$-matrices?

Solution to Matrix multiplication
The recursive equation is $T(n) = 22 \cdot T(n/3) + O(n^2)$. The master theorem gives the solution $T(n) = O(n^{\log_3 22}) = O(n^{2.814})$. □

Complex multiplication If we multiply two complex numbers $a+bi$ and $c+di$ in the standard way, it requires four multiplications and two additions of real numbers. Since multiplications are more expensive than additions (and subtractions), it pays to minimize the number of multiplications if one would allow large numbers. Find an algorithm that uses only three multiplications (but more additions) to multiply two complex numbers.

Solution to Complex multiplication
Exercise. □

Majority Consider an array A of n elements (say integers). Construct and analyze an algorithm to determine whether any element of the array A is in majority, namely, it occurs in A at least $n/2$ times. If this is the case, return it. The algorithm will be a recursive one and will have time complexity $O(n \log n)$. The only operation you are allowed to use on the elements of A is $=$ (equality test). Moreover, there is no order relationship between the elements.

Solution to Majority
If there is a majority element, it must be of at least one half of the array elements.
Recursive view: Check majority recursively in the left and right half of the array and then count how many times the half array majority elements are present in the entire array. If any elements are in total majority return it.

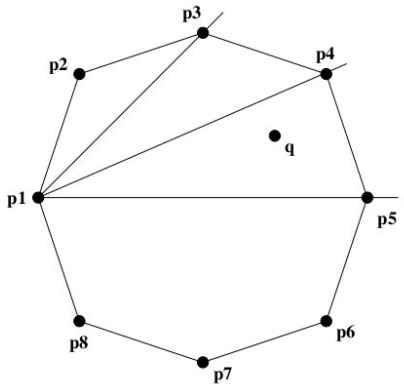
Majority($A[1..n]$) =
    if $n = 1$ **then return** $A[1]$
    $m \leftarrow \lceil n/2 \rceil$
    $v \leftarrow$ Majority($A[1..m-1]$)
    $h \leftarrow$ Majority($A[m..n]$)
    if $v = h$ **then return** v
    $vn \leftarrow 0; hn \leftarrow 0$
    for $i \leftarrow 1$ **to** n **do**
        if $A[i] = v$ **then** $vn \leftarrow vn + 1$
        else if $A[i] = h$ **then** $hn \leftarrow hn + 1$
    if $vn \geq m$ **then return** v
    if $hn \geq m$ **then return** h
    else return NULL

Time complexity: Each recursive call will half the array and perform $O(n)$ operations. Using the master theorem we get $O(n \log n)$ time complexity.

□

Inside or outside? Let P be a convex n-angle polygon described as an array of angles $p_1, p_2, \ldots p_n$ in cyclic order. Construct an algorithm that computes whether a given point q is inside the polygon P. The algorithm will run in time $O(\log n)$ in worst case.

Solution to Inside or outside?
We use an interval halving search to exclude the half of the remaining polygon each time, until only one triangle remains. Then we can easily (with a constant number of comparisons) to determine whether q lies in P. See Figure 1.



Figur 1: A convex polygon and the lines that the algorithm uses to halve it.

$InsideConvex(P, q, l, u) =$
  if $u = l + 1$ **then** /* a triangel */
    choose a point q' outside the triangle p_1–p_l–p_u
    if line q–q' cut exactly one of the edges of the triangle **then**
      return inside
    else
      return outside
  else
    $mid \leftarrow \lceil \frac{l+u}{2} \rceil$
    if q is on the same side of line p_1–p_{mid} as p_{mid+1} **then**
      return $InsideConvex(P, q, mid, u)$
    else
      return $InsideConvex(P, q, l, mid)$

The algorithm starts with $InsideConvex(P, q, 2, n)$.

If we assume that $InsideConvex(P, q, 2, n)$ take time $T(n)$ we get the recursive equation

$$T(n) = T\left(\frac{n}{2}\right) + c$$

which has solution $c \log n$. Thus, the time complexity is $T(n) \in O(\log n)$. □
