Homework 3 by Timofei Podlorytov

3.1

Considering the following pairs of functions f and g, show for each pair whether or not it belongs to each of the relations $f \in \Theta(g)$, $f \in O(g)$, $f \in o(g)$, $f \in \Omega(g)$, $f \in \omega(g)$, $g \in \Theta(f)$, $g \in O(f)$, $g \in o(f)$, $g \in \Omega(f)$, or $g \in \omega(f)$.

(a) (2 points) f (n) = 9n and g(n) = 5n^3 ,
(b) (2 points) f (n) = 9n^0.8 + 2n^0.3 + 14 log n and g(n) =√n,
(c) (2 points) f (n) = n^2 / log n and g(n) = n log n,
(d) (2 points) f (n) = (log(3n))^3 and g(n) = 9 log n.

(a)
lets's find the limit of f(n)/g(n) and g(n)/f(9n) and use that in order to determine their relationtship. Firstly, we need to mention that for positive n the functions are non negative.
$\lim_{n\to\infty} \frac{5n^3}{9n} = \lim_{n\to\infty} \frac{5n^2}{9} = \infty$- we can conclude that $f \in \omega(g)$ and $g \in o(f)$ (according to the definitions). Also, since ω o are defined for any constant c we can find $n_0$whie for O and Ω we have that such a constant must exist we can conclude that the following statement is true as well

plus it follows from the limit being >0 (for Ω) and for O the limit is 0 which is smaller than infinity

$g \in O(f)$, $f \in \Omega(g)$

since the limits we can get are either infinity or 0 we can conclude that they are not tight bounds of one another.
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ and $\lim_{n\to\infty} \frac{g(n)}{n(n)} = 0$

$f \in \Theta(g)$, $f \in O(g)$, $f \in o(g)$, $f \in \Omega(g)$, $f \in \omega(g)$
, $g \in \Theta(f)$, $g \in O(f)$,$g \in o(f)$, $g \in \Omega(f)$, or $g \in \omega(f)$.

(b)
we follow a similar logic here as well.
$\lim_{n\to\infty} \frac{9n^{0.8}+2n^{0.3}+14\log(n)}{\sqrt{n}} = \lim_{n\to\infty} \frac{9n^{0.3}+2n^{-0.2}+14\log(n)/\sqrt{n}}{1} = \lim_{n\to\infty} (9n^{0.3}+2n^{-0.2}) + \lim_{n\to\infty} \frac{14\log(n)}{\sqrt{n}} = ¿$
I used Hopital's rule for one of the elements of the sum
$\lim_{n\to\infty} (9n^{0.3}+2/\sqrt[5]{x}) + \lim_{n\to\infty} \frac{14*\sqrt{n}}{\ln(2)*n*0.5} = \lim_{n\to\infty} (9\infty+0) + \lim_{n\to\infty} \frac{14}{\ln(2)*\sqrt{n}*0.5} = \infty+0 = \infty$
we conclude similarly to the previous sub task that $f \in \omega(g)$ and $g \in o(f)$ and $g \in O(f)$, $f \in \Omega(g)$ since the limit we got is infinity
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ and $\lim_{n\to\infty} \frac{g(n)}{n(n)} = 0$

$f \in \Theta(g)$, $f \in O(g)$, $f \in o(g)$, $f \in \Omega(g)$, $f \in \omega(g)$
, $g \in \Theta(f)$, $g \in O(f)$,$g \in o(f)$, $g \in \Omega(f)$, or $g \in \omega(f)$.

(c)

Let's try to find a limit again:
f (n) = n^2 / log n and g(n) = n log n

$\lim_{n\to\infty} \frac{n^2}{\log(n)} \frac{1}{n\log(n)} = \lim_{n\to\infty} \frac{n}{\log(n)^2}$ we use Hopital's rule to determine the limit

$\lim_{n\to\infty} \frac{\ln(2)n}{2\log(n)} = \lim_{n\to\infty} \frac{\ln(2)^2 n}{2} = \infty$ we get infinity and conclude the following relations f $\in$ ω(g) and g $\in$ o(f) and g $\in$ O(f), f $\in$ Ω(g)

We got the following results:
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ and $\lim_{n\to\infty} \frac{g(n)}{n(n)} = 0$
f $\in$ Θ(g), f $\in$ O(g), f $\in$ o(g), f $\in$ Ω(g), f $\in$ ω(g)
, g $\in$ Θ(f), g $\in$ O(f),g $\in$ o(f), g $\in$ Ω(f), or g $\in$ ω(f).

(d)
In order to determine how the 2 functions relate to each other we find the limit of their division like before

$$\lim_{n\to\infty} \frac{(\log(3n))^3}{9\log(n)} = \lim_{n\to\infty} \frac{3(\log(3n))^2 * \ln(2) * n}{\ln(2) * 3n * 9} = \lim_{n\to\infty} \frac{(\log(3n))^2}{9} = \infty$$

as before that leads to the following expressions
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ and $\lim_{n\to\infty} \frac{g(n)}{n(n)} = 0$ which give us f $\in$ ω(g) and g $\in$ o(f) and g $\in$ O(f), f $\in$ Ω(g)

f $\in$ Θ(g), f $\in$ O(g), f $\in$ o(g), f $\in$ Ω(g), f $\in$ ω(g)
, g $\in$ Θ(f), g $\in$ O(f),g $\in$ o(f), g $\in$ Ω(f), or g $\in$ ω(f).

3.2

Code as well as comments are in the cpp file but I duplicated some answers here as well.
Most explanations are there
a)
The implementation of the algorithm as a function is written here:

```
void selectSort(int* array, int len){
        for(int i=0; i<len-1;i++){
                int min=array[i+1];//we need to compare the array to smth
                int pl= i+1;//we set the value of the next as a min
                for(int j=i+2;j<len;j++){//we search for the minimum value
                        if(array[j]<min){
                                min=array[j];
                                pl=j;//we save the place as well
                        }
                }
                if(min<array[i]){
```

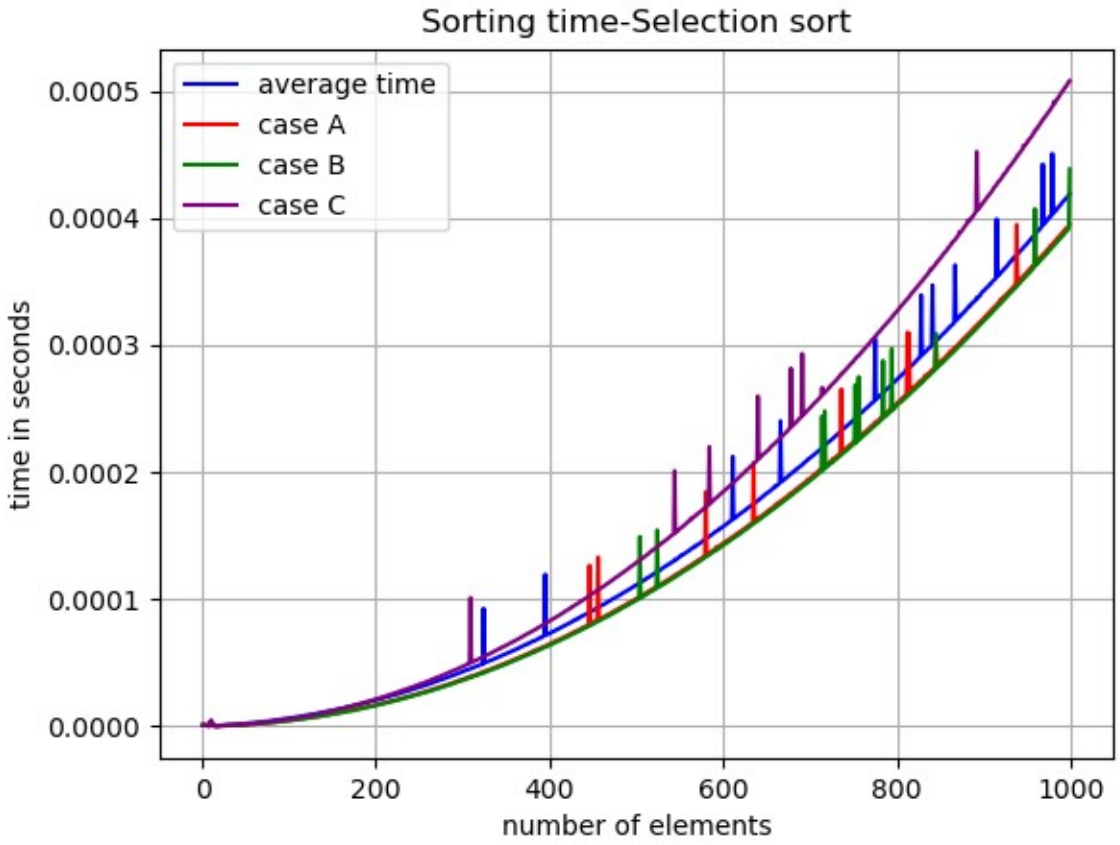
```
                    array[pl]=array[i];
                    array[i]=min;//swap
                }
            }
}
```

b)
the loop invariant:
as an invariant we have that all the elements up to i(included) are sorted. Which can be easily seen in the function that sorts while showing the steps.

```
void selectSortShow(int* array, int len){//look at acutal cpp file
```

Case B is just increasing order that requires 0 swaps. Case A, which is the worst case, has 2 options(look task d). The execution time in my code varies due to if statements and their execution. According to the task it has to have the most swaps. We want to swap almost every single element which is n.
Let's take an array 2 3 4 1→ 1 3 4 2→ 1 2 4 3→ 1 2 3 4 we have n-1 swaps here. Every iteration we have the smallest element(from the unsorted part) on the end and thus have to swap. So, the sorted order with only the last element being the smallest in the array is the worst case by the number of swaps

Now we can take a decreasing order 4 3 2 1 → 1 3 2 4 → 1 2 3 4 we have n/2 swaps which is less. Basically we mirror the array and the operation is finished when we get to the middle. So, with respect to the number of swaps it's not the worst case.
c)
my codes run time does not seemingly depend on the order of the elements since I used a for loop which has to go through all elements and always performs a swap. But it depends on if statements. And surprisingly case A is not the worst there. By execution time the decreasing order wins as there we go through the inner loop if statement and assign the minimum value for each iteration and thus cause for time to be spent running, even though it has less swaps than case A. This case I called case C and plotted on the graph as well.
d)



I wrote a short python code that plots the measurements. I ran it up till size of array 1000 and took at least 30 samples.

e)
Here we can see that A and B perform almost the same, while the decreasing order stands out as the winner in worst time due to the reasons mentioned above. The peaks could be explained by imperfections of how cpu runs. The resulting curves appear quadratic and take the shape of parabola. Their asymptotic behavior could be summed up with all cases going to infinity when n goes to infinity. All parabolas have their vertex in n=0. → f(n)=const*n^2 approximately const = time/n^2

for A, B and random const≈4*10^(-4)*10^(-6)=4*10^(-10)
for C it's about 5*10^(-10) which is larger and implies faster growth.