Problem 1.1

(a) $f(n) = 3n$ and $g(n) = n^3$

$0 \leqslant f(n) \leqslant cg(n)$, $0 \leqslant cf(n) \leqslant g(n)$ for some $+c$ and $\forall n \geqslant n_0$

Let $c=2$; $cf(n)=2*3n=6n$; $cg(n)=2n^3$.

If $n=2$, $g(n)=8$, $cf(n)=12$. So, $g(n) \leqslant cf(n)$

If $n=1$, $f(n)=3$, $cg(n)=1$. So, $cg(n) \leqslant f(n)$

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{3n}{n^3}\right) = \lim_{n\to\infty}\left(\frac{3}{n^2}\right) = 0$$

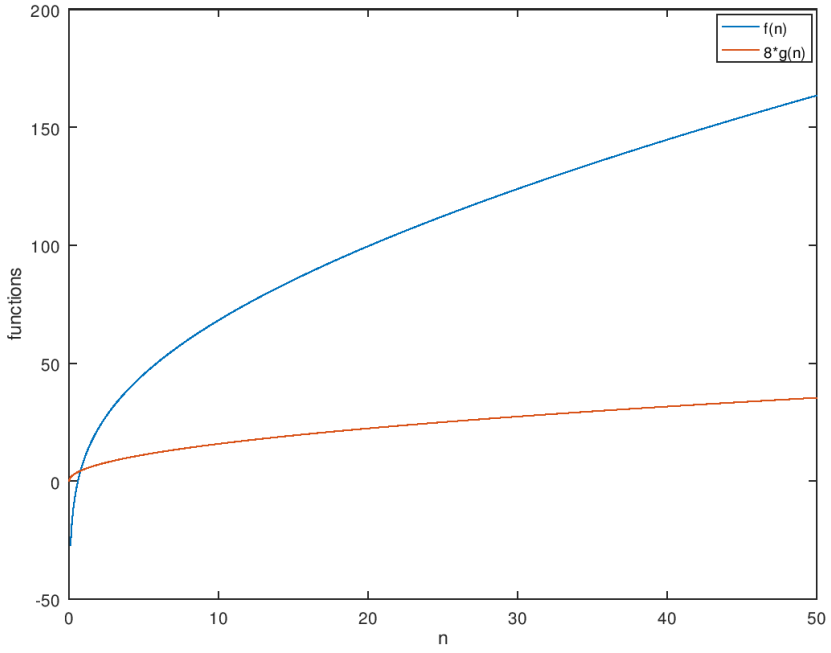
$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{n^3}{3n}\right) = \lim_{n\to\infty}\left(\frac{n^2}{3}\right) = \infty$$

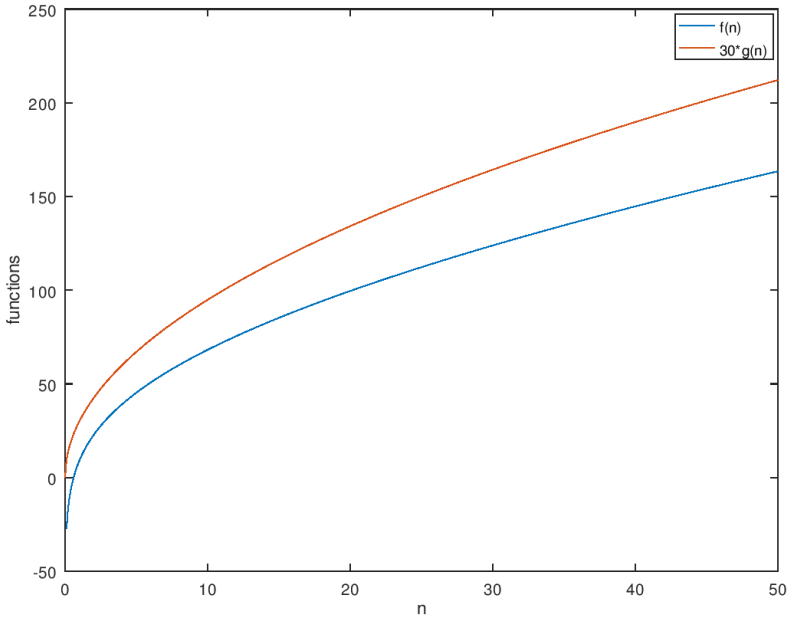
So, $f(n) = o(g(n))$, and $g(n) = \omega(f(n))$

Also, $f(n)= O(g(n))$, and $g(n)= \Omega(f(n))$

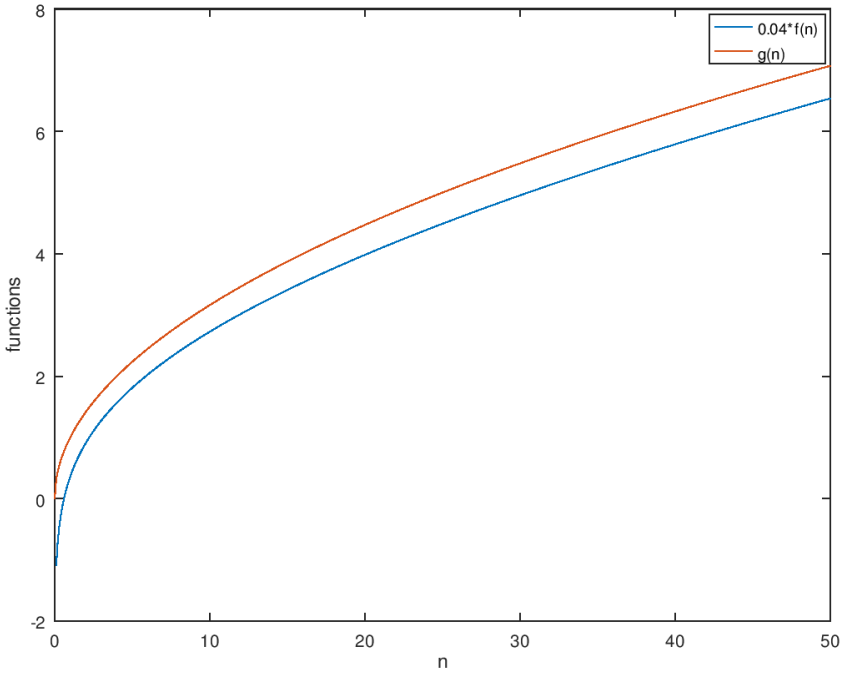
(b) $f(n) = 7n^{0.7} + 2n^{0.2} + 13\log(n)$, $g(n) = \text{sqrt}(n) = n^{0.5}$

$0 \leqslant cg(n) \leqslant f(n)$ for some $+c$ and $n \geqslant n_0$

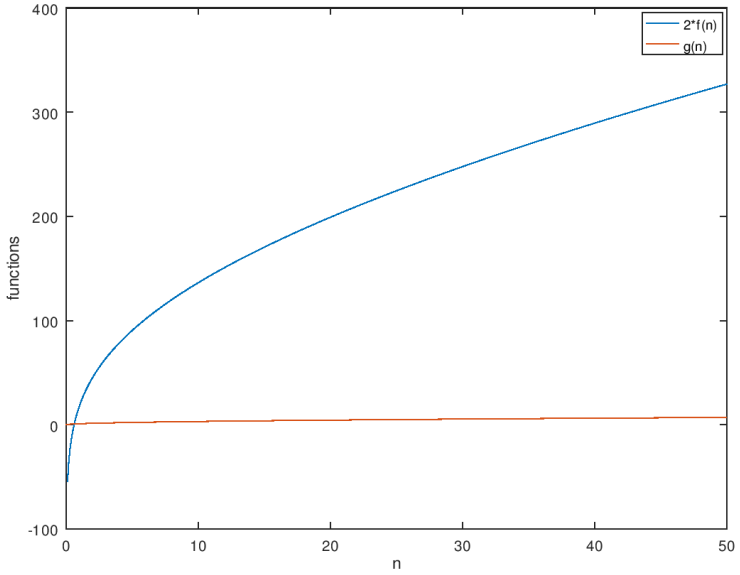




In the figure above, f(n) can become larger than 30g(n) if n grows continuously.



In the figure above, 0.04f(n) will become larger than g(n) if n grows continuously.



$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{7n^{0.7}+2n^{0.2}+13\log(n)}{n^{0.5}}\right) = \infty$$

$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{n^{0.5}}{7n^{0.7}+2n^{0.2}+13\log(n)}\right) = 0$$

So, g(n) = O(f(n)), and f(n) = Ω(g(n))

Also, g(n)= o(f(n)), and f(n)= ω(g(n))

(c) f(n) = n^2/log(n), g(n) = n log(n)

$0 \leqslant cg(n) \leqslant f(n)$ for some +c and $\forall n \geqslant n_0$

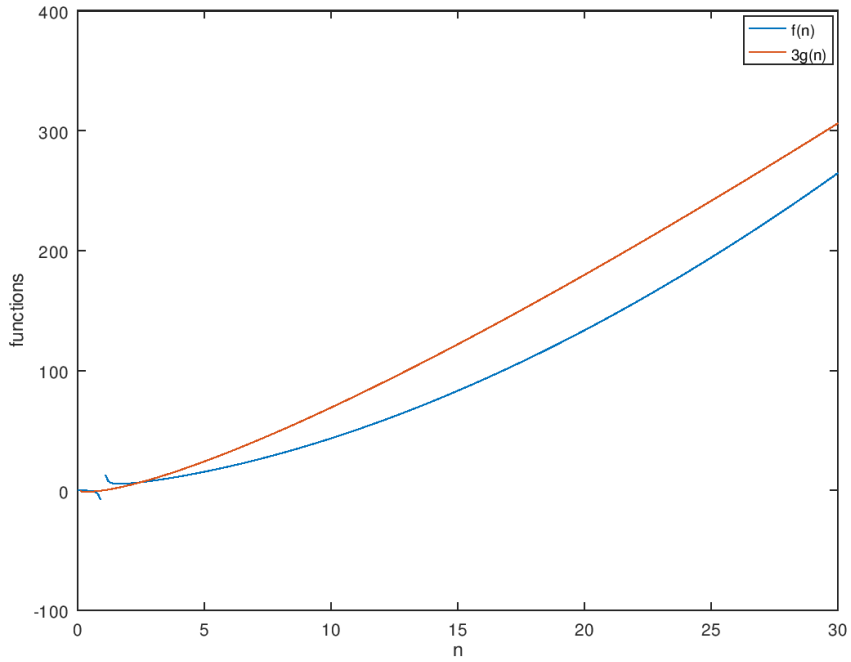
$0 \leqslant g(n) \leqslant cf(n)$ for some +c and $\forall n \geqslant n_0$

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{n^2/\log(n)}{n\log(n)}\right) = \lim_{n\to\infty}\left(\frac{n}{(\log(n))^2}\right) = \infty$$

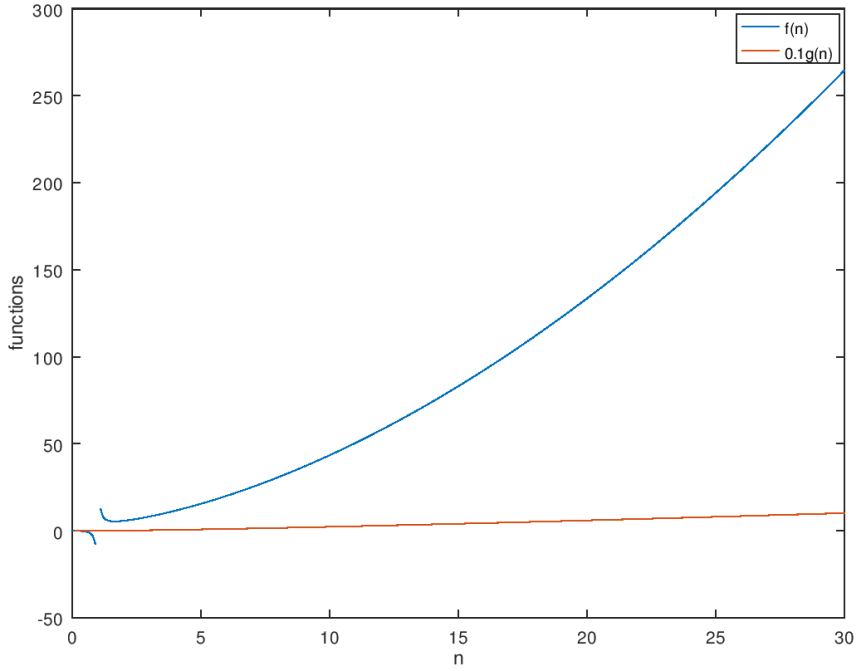
$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = \lim_{n\to\infty}\left(\frac{n\log(n)}{n^2/\log(n)}\right) = \lim_{n\to\infty}\left(\frac{(\log(n))^2}{n}\right) = 0$$

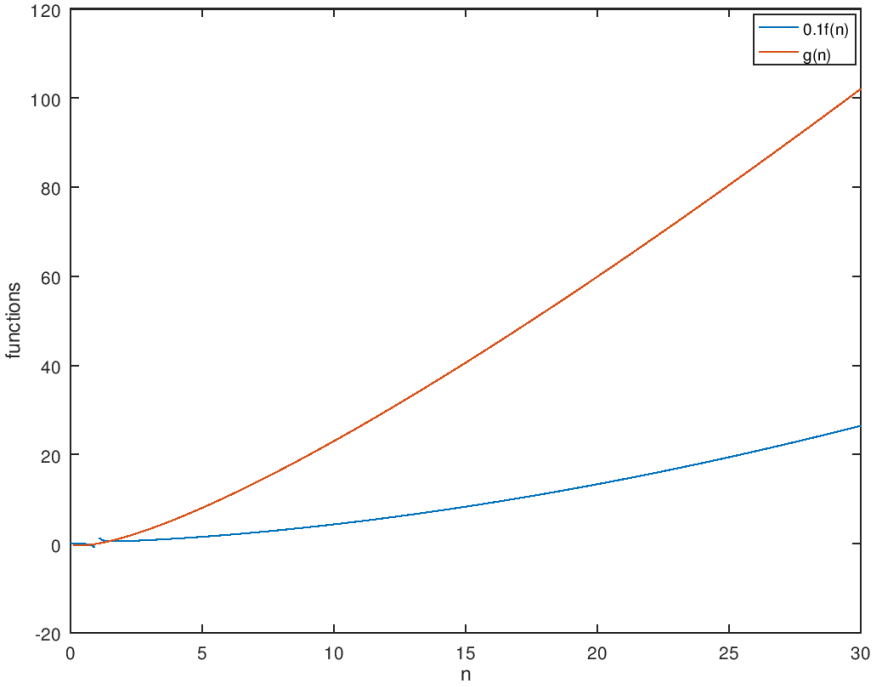
So, g(n) = O(f(n)) and f(n) = Ω(g(n))

Also, g(n)= o(f(n)), and f(n)= ω(g(n))

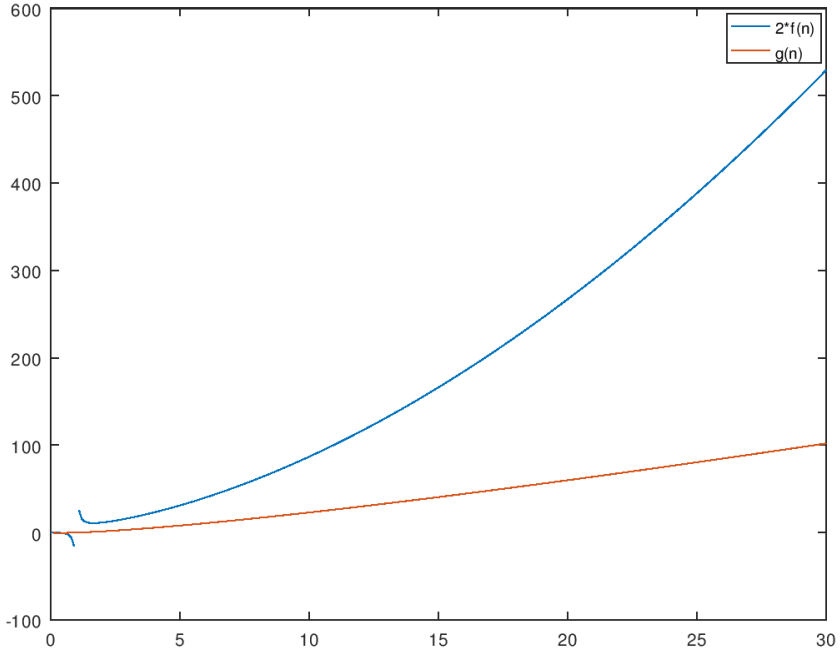


In the figure above, f(n) can become larger than 3g(n) if n grows continuously.





In the figure above, 0.1f(n) might become larger than g(n) if n grows continuously.

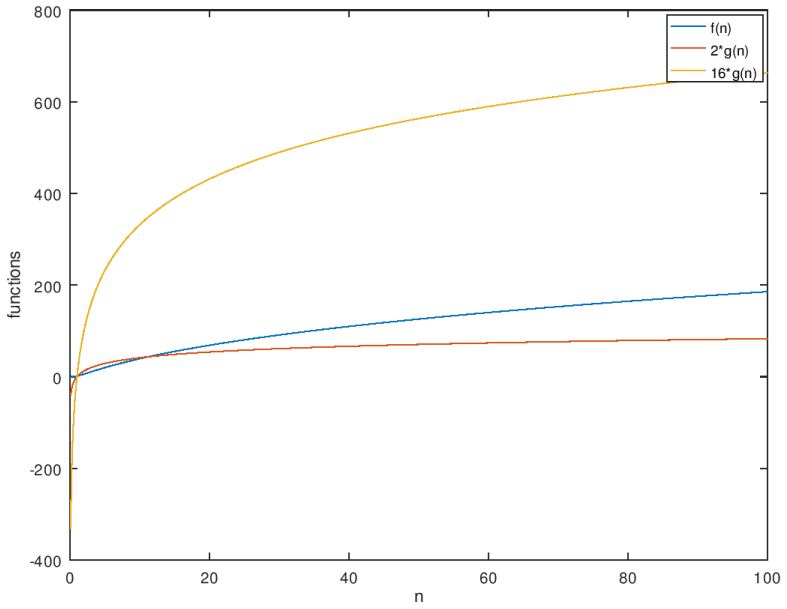


X-label: n, Y-label: functions

(d) $f(n) = (\log(3n))^3$, $g(n) = 9 \log(n)$

Let $c_1 = 4$, $c_2 = 16$; $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$, $\forall n \geqslant n_0$,

So $f(n) = \theta(g(n))$



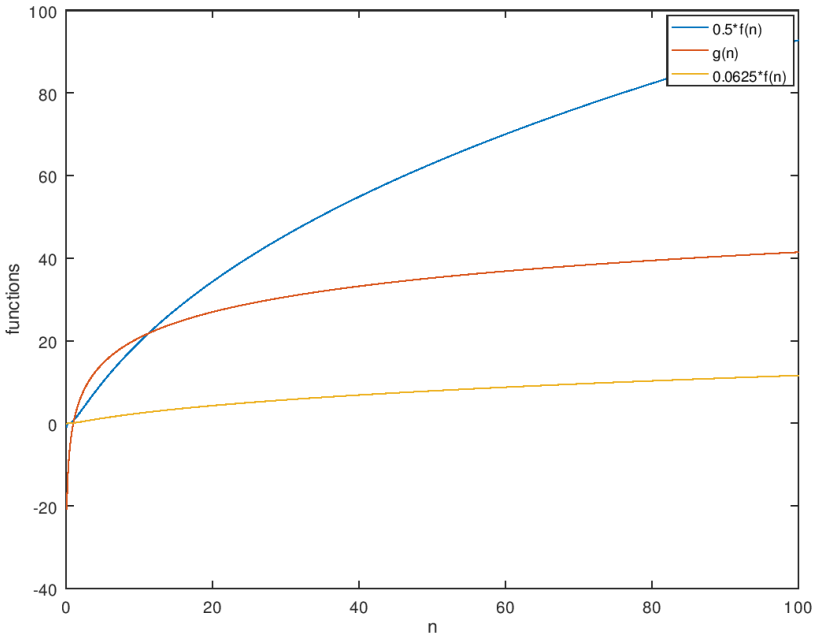
Since $f(n) = \theta\big(g(n)\big)$ holds, $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ is true.

$c_1 g(n) \leqslant f(n) \rightarrow g(n) \leqslant (1/c_1)\, f(n)$.

$f(n) \leqslant c_2 g(n) \rightarrow (1/c_2)\, f(n) \leqslant g(n)$

$(1/c_2)\, f(n) \leqslant g(n) \leqslant (1/c_1)\, f(n)$. So, $g(n) = \theta\big(f(n)\big)$ is true.

Also, g(n) = O(f(n)), f(n) = Ω(g(n)), and f(n)= O(g(n)), g(n)= Ω(f(n)) hold.



Problem 1.2

(a) The user is asked to type in the number of elements randomly, and a random array with the size of input will be sorted with the function below. The full code is Selection Sort a.c file.

```
swap(a, b)
        int temp = *a;
        int temp = *a;
        *b = temp;
SelectionSort(A,n)
      for (i=0 to n-1)
      min=i
          for(j=i+1 to n)
              if(A(j)<A(min))
                  min=j
           if(min!=i)
                swap(A(i),A(min))
```

(b) Loop Invariant

If the number of elements is 1, the arrays will contain only one element, and that element will remain as the minimum element without any changes.

If no.of elements > 1, the index of minimum element is i in the outer loop, and the loop will work between i=[0..sizeof(array)-1].. Then, the inner loop will start from (i+1)th element to check whether there is a smaller element on the right side of i^{th} element; the loop will continue between [1..sizeof(array)]. If yes, the current min element and the new smaller element will be swapped in the outer loop. If min=i is still true after the inner loop, both i and min will be incremented.

After the outer loop has looped once, all elements in [0..(i-1)] indexes of the array are smaller than or equal to the remaining elements [i..(n-1)] indexes of the array. Also, in the inner loop, all elements in [1..(j-1)] indexes are smaller or equal to the element in array[min]. When both outer loop is finished, all elements in the array will be sorted successfully.

For example, we have an array with values [5,12,3,22,8] where n=5

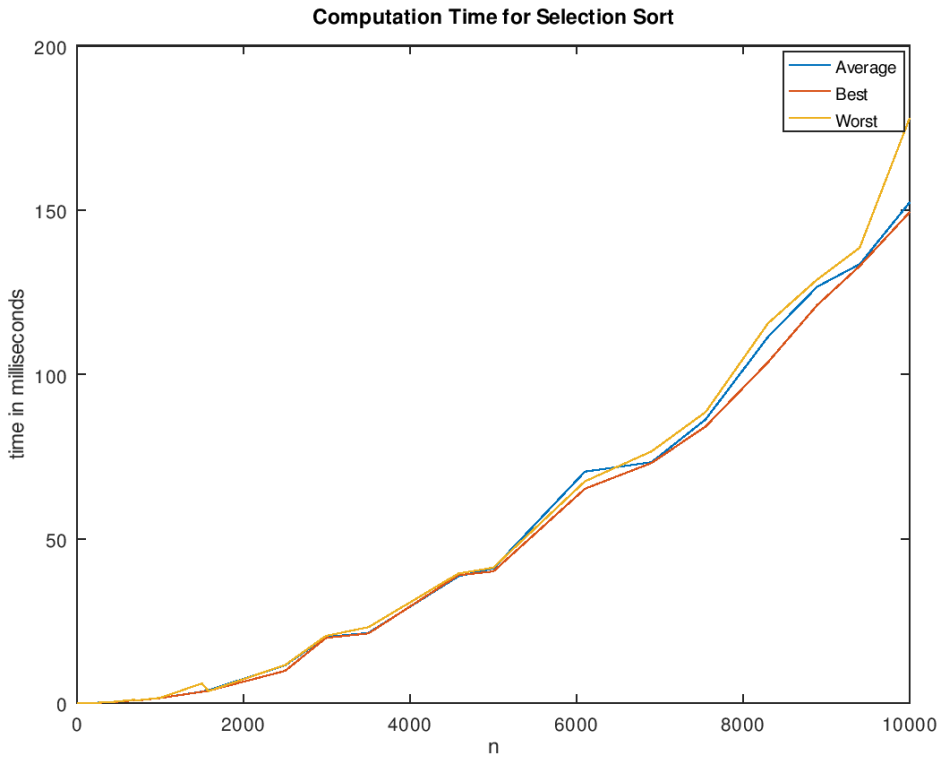
swap(a, b)	
int temp = *a;	
int temp = *a;	
*b = temp;	
SelectionSort(A,n)	

for (i=0 to n-1)	i= ~~0 1 2 3 4~~
min=i	min= ~~0 1 2 3 4~~
for(j=i+1 to n)	j= ~~1 2 3 4 5~~ j= ~~2 3 4 5~~ j= ~~3 4 5~~ j= ~~4 5~~
if(A(j)<A(min))	i=0→ ~~12!<5~~ ~~3<5~~ ~~3<22~~ ~~3<16~~ i=1→ ~~12!<5~~ ~~22!<5~~ ~~16!<5~~ i=2→ ~~22!<12~~ ~~8<12~~ i=3→ ~~12<22~~ i=4→A(5) does not exist, so the inner loop stops and finishes.
min=j	i=0→min= ~~0 2~~ i=1→min= ~~1~~ i=2→min= ~~2 5~~ i=3→min= ~~3 4~~
if(min!=i)	i=0→~~False~~ ~~True~~ i=1→~~False~~ i=2→~~False~~ ~~True~~ i=3→False(min=i=4)
swap(A(i),A(min))	i=0→swap(5,3)→[3,5,12,22,8] i=1→no swap made i=2→swap(12,8)→[3,5,8,22,12] i=3→swap(22,12)→[3,5,8,12,22] i=4→no swap made and the outer loop finishes

So, the numbers are swapped successfully in an incenting order. [5,12,3,22,8]→ [3,5,8,12,22]

(c) The random numbers for the array to be sorted are generated by rand() function in C language. The final sorted array will be copied to a new array in the same order, and that new array will be used to measure the best case result. The final sorted array will be copied in a reverse order to another new array, and that array will be sued to measure the worst case result. The details are included in Selection Sort c.c file.

(d) The input n is increased, and the time taken for random case, best case and worst case are saved in text files. The algorithm is run with same n multiple times to take the average case. Afterwards, the average case is calculated in a spreadsheet, and n, average case, best case and worst case results are exported to a new text file. Then, a graph is plotted by using the results from that new text file in Matlab. The full code is written in Selection Sort d.c file.



(e) The algorithm of selection sort contains two nested loops. Normally, the computation time complexity for two nested loops is $O(n^2)$ since the maximum time taken (in the inner loop) is the $\sum_{j=1}^{n-1}(n-j) = \frac{n(n-1)}{2}$. So, the cost*time taken for the inner loop($c*\frac{n(n-1)}{2}$) will be the maximum. Also, the graphs for average case, best case and worst case are close to each other. Thus, time complexity for selection sort belongs to $O(n^2)$.