1 QuickSort

See the source code file `quickSort.cpp` and the tests given in `tests.cpp`. No written response is needed for this part of the lab.

2 Big-O Proofs

Problem 1. Show that $8n^3 + 7n^2 - 12$ is $O(n^3)$.

Proof. Assume $n > 0$ and $n \in \mathbb{Z}$. Observe

$$\begin{cases} 8n^3 \leq 8n^3 \\ 7n^2 \leq 7n^3 \\ -12 \leq 0 \end{cases}$$

Notice $8n^3 + 7n^2 - 12 \leq 8n^3 + 7n^3 = \underbrace{15}_{c} n^3$. Therefore, $8n^3 + 7n^2 - 12$ is $O(n^3)$ when $n_0 = 1, c = 15$.

$\square$

Problem 2. Show that $6n^2 - n + 4$ is $O(n^2)$.

Proof. Assume $n > 0$ and $n \in \mathbb{Z}$. Observe

$$\begin{cases} 6n^2 \leq 6n^2 \\ -n \leq 0 \\ 4 \leq 4n^2 \end{cases}$$

Notice $6n^2 - n + 4 \leq 6n^2 + 4n^2 = \underbrace{10}_{c} n^2$. Therefore, $6n^2 - n + 4$ is $O(n^2)$ when $n_0 = 1, c = 10$.

$\square$

3 Mystery Functions

A

```
Function fnA(n):
    For i In 1 To n/2:
        Set a To i
    EndFor
EndFunction
```

We claim that `fnA(n)` is $O(n)$. Inside the first loop there is only one step of constant time, and the bigger loop has $\frac{n}{2}$ steps. Thus, the program can be generalized as $O(n)$.

B

```
Function fnB(n):
    For i In 1 to n:
        For j In 1 to n:
            Set a To i
        EndFor
    EndFor
EndFunction
```

We claim that `fnB(n)` is $O(n^2)$. Observe the program consists of a nested loop, while the outer loop executes n times, the inner loop executes n times. The command inside the inner loop is only one step of constant time. Thus, the outer loop executes n times over the inner loop that runs for n times, which conforms with $O(n^2)$.

C

```
Function fnC(n):
    For i In 1 to n:
        Set j To 1
        While j < n:
            Set j To j*2
        EndWhile
    EndFor
EndFunction
```

We claim that `fnC(n)` is $O(n \log n)$. The program is complicated by a for loop that runs n times and a while loop inside. We interpret that the while loop exits after $\log_2 n$ times, because each iteration the variable `j` approaches `n` doubly faster than the last iteration. As `i` gets larger linearly by virtue of the for loop, it takes less time for the while loop to end.

D

```
Function fnD(n):
    For i In 1 to n*n:
        For j In 1 to n*n:
            Set a To j
        EndFor
    EndFor
EndFunction
```

We claim that `fnD(n)` is $O(n^4)$. The program is quite akin to `fnB(n)`, while the only difference is that both the inner and the outer loop takes n^2 steps instead. Consequently, the overall execution would take a $n^2 \cdot n^2 = n^4$ time.

E

```
Function fnE(n):
    For j In 1 To 4:
        Set a To i
    EndFor
EndFunction
```

We claim that `fnE(n)` is $O(1)$. Whatever n is does not affect the running time of the program, given that n is not involved in the execution of the for loop, which indicates that the program is of constant time.

F

```
Function fnF(n):
    Set i To 0
    While i < n*n*n:
        Set i To i + 1
    EndWhile
EndFunction
```

We claim that `fnF(n)` is $O(n^3)$. Since `i` always starts with a value of 0, the while loop should always take `n*n*n` folds.

We claim the sorted order of the functions from fastest to slowests is

$$E < A < C < B < F < D$$