Danyl Fernandes
2020012004 (72)

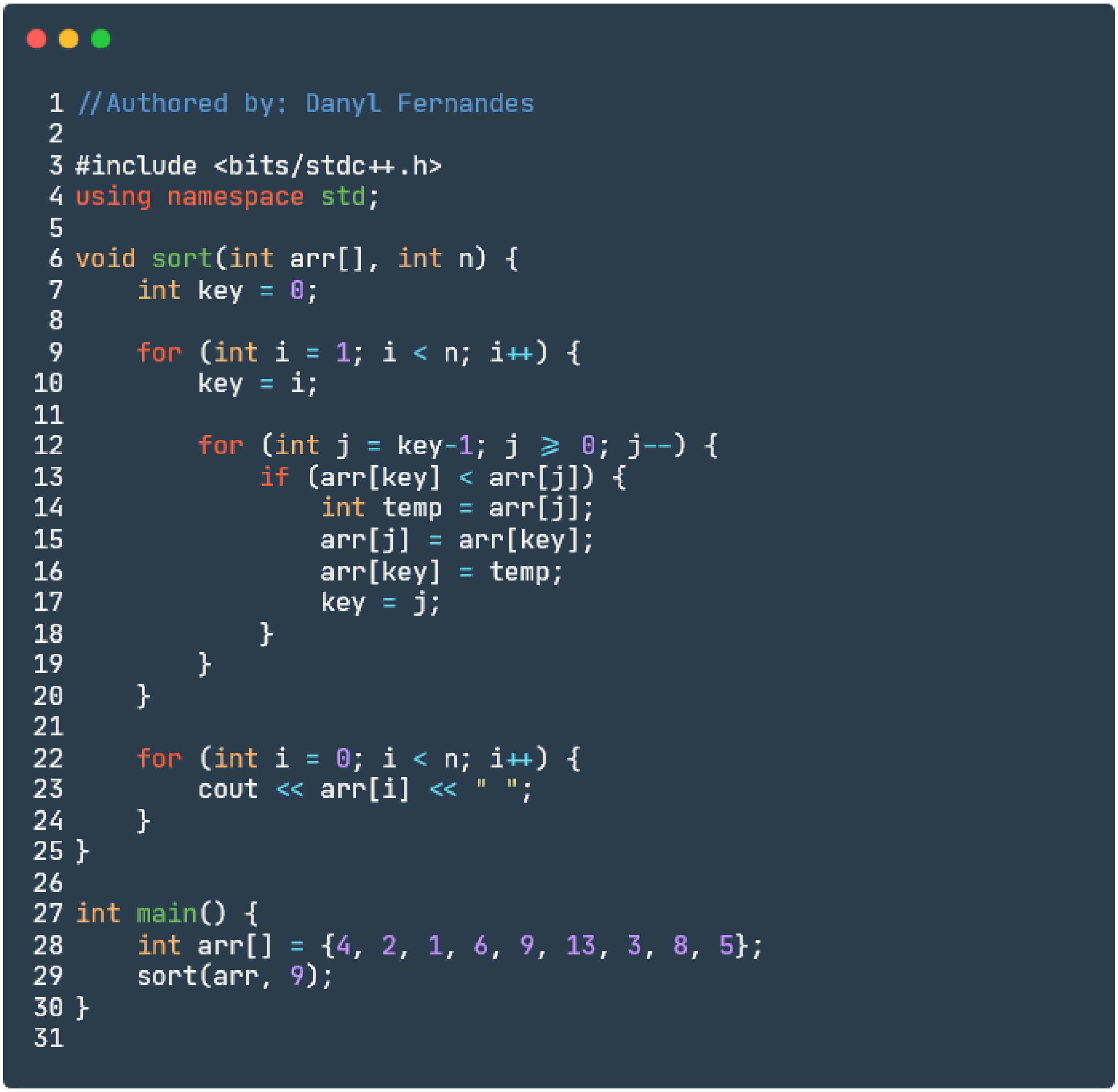
11-04-2021

AOA Experiment 1

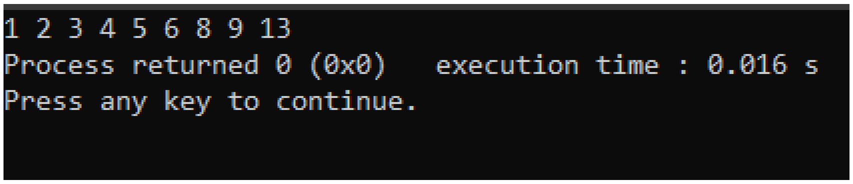
Aim:

To implement & analyze Insertion Sort Algorithm and compare its complexity with Selection Sort:

Implementation:



```cpp
//Authored by: Danyl Fernandes

#include <bits/stdc++.h>
using namespace std;

void sort(int arr[], int n) {
    int key = 0;

    for (int i = 1; i < n; i++) {
        key = i;

        for (int j = key-1; j >= 0; j--) {
            if (arr[key] < arr[j]) {
                int temp = arr[j];
                arr[j] = arr[key];
                arr[key] = temp;
                key = j;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
}

int main() {
    int arr[] = {4, 2, 1, 6, 9, 13, 3, 8, 5};
    sort(arr, 9);
}
```



```
1 2 3 4 5 6 8 9 13
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Danyl Fernandes
2020012004 (72)

Exp 01

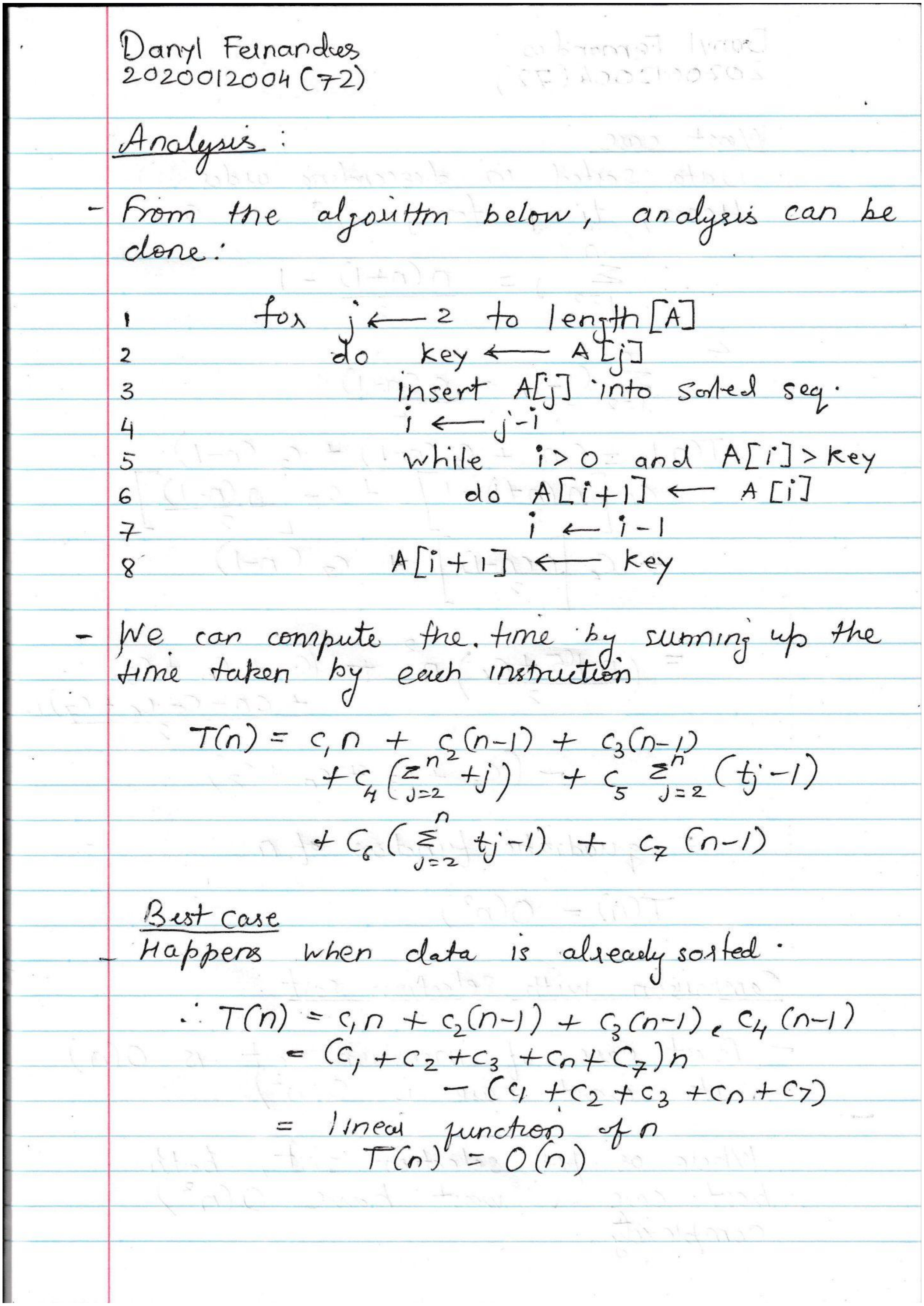
Theory

- Insertion sort uses the analogy of sorting cards by hand. One card is removed at a time from the deck & it is inserted a the correct location in thand itself. Upcoming cards are processed in the sam way

- To insert a new card, all the cards in hand having value larger than new card are shifted on the right side by one. New card is inserted on space created after moving some 'k' cards on the righ side.

- Insertion sort is an in place algorithm. It does not require extra memory. Sortin is done in the input array itself. In iteration 'k', first 'k' elements are always sorted.

- Running time is the number of steps required to. solve the problem on RAM model. Each instruction may take differen amount of time.



Danyl Fernandes
2020012004 (72)

Analysis :

- From the algorithm below, analysis can be done:

```
for j ← 2 to length[A]
    do key ← A[j]
        insert A[j] into sorted seq.
        i ← j-1
        while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                i ← i-1
        A[i+1] ← key
```

- We can compute the time by summing up the time taken by each instruction

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\sum_{j=2}^{n} t_j\right) + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6\left(\sum_{j=2}^{n} t_j - 1\right) + c_7(n-1)$$

Best case

- Happens when data is already sorted.

$$\therefore T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1)$$
$$= (c_1 + c_2 + c_3 + c_n + c_7)n - (c_1 + c_2 + c_3 + c_n + c_7)$$

= linear function of n

$T(n) = O(n)$

Danyl Femandues
2020012004 (72)

Worst case:

Data sorted in descending order

Hence, $t_j = j$ for $j = 2, 3 \cdots n$

$$\therefore \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

&

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\therefore T(n) = c_1 n + c_2 (n-1) + c_3 (n-1)$$
$$c_n \left[\frac{n(n+1)}{2} - 1\right] + c_5 \left[\frac{n(n-1)}{2}\right]$$
$$+ c_6 \left[\frac{n(n-1)}{2}\right] + c_7 (n-1)$$

$$= \left(\frac{n + c5 + c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{cn - c_5 - c_6 + c_7}{2}\right) n$$
$$- (c_2 + c_3 + c_n + c_7)$$

= quadratic function of n

$$T(n) = O(n^2)$$

Comparison with selection sort:

- Best case of insertion sort is $O(n)$ & its worst case is $O(n^2)$

Where as for selection sort, both best case & worst have $O(n^2)$ complexity.

Danyl Fernandes
2020012004(72)

- Hence, we concur that insertion sort is relatively faster than Selection Sort.

Example :

a) 5 2 4 6 1 3

b) 2 5 4 6 1 3

c) 2 4 5 6 1 3

d) 2 4 5 6 1 3

e) 1 2 4 5 6 3

f) 1 2 3 4 5 6

Conclusion : We implemented the insertion sort, analyzed it & successfully compared its complexity with that of Selection sort.