Swapnil Paxini (V00861672) 24/02/2017

CSC 225 Assignment 2

Q1) Size of array = N

$$f(N) = \begin{cases} N + c & , \text{ Strategy 1} \\ 2N & , \text{ Strategy 2} \end{cases}$$

∴ cost of regular push = 1
cost of special push (with extending) = $f(N) + N + 1$

Using Strategy 1:
Let each phase be between two times you need to extend the array:

eg. c = 3
1st push = 3(1) + 0 + 1
= 4
2nd push = 1
3rd push = 1

} phase 1, cost = 6

Similarly phase 2 cost is 12. Total phases = $\frac{N}{c}$

By this we can get a relation that:

$$\text{Total cost} = \sum_{i=1}^{\frac{N}{c}} 2ic$$

Doing the same for Strategy 2:

eg. :

phase 1 : 1

phase 2 : 2 4

phase 3 : 8

phase 4 : 16

$\therefore$ number of phases : $\log_2 N$

$$\therefore \text{Total cost} = \sum_{i=1}^{\log_2 N} 2^i$$

Total cost for strategy 1 : $\sum_{i=1}^{\frac{N}{c}} 2ic$

" " " " 2 : $\sum_{i=1}^{\log N} 2^i$

Strategy 1 : $\therefore \sum_{i=1}^{\frac{N}{c}} 2ic = 2c \sum_{i=1}^{\frac{N}{c}} i = 2c\left(\frac{\frac{N}{c}\left(\frac{N}{c}+1\right)}{2}\right) = N\left(\frac{N}{c}+1\right)$

$$= \frac{N^2}{c} + N$$

$$= O(N^2)$$

Strategy 2 : $\sum_{i=1}^{\log N} 2^i = 2^{\log_2 N + 1} - 1$

$$= 2N - 1$$

$$= O(N)$$

$\therefore$ Strategy 2 is faster.

Q2) When using insertion sort, in essence you remove all the inversions in the array one by one. Therefore, the number of times the while loop within the for loop of insertion executes is equal to the number of inversions in the array.

Therefore the running time of Insertion sort with n key and k inversions is:

$\underline{O(n+k)}$ as n is number of time the for loop executes and k is the # of inversions as well as the number of comparisons made within the for loop.

- In the worst case scenario, when the array is reverse ordered, the number of inversion will $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$

$\therefore O(n+k) = O(n+n^2) = \underline{O(n^2)}$ ← worst case

Q3) In order to make it $O(n \log n)$, we need implement a modified merge sort with a merge method that counts the inversions.

```
countInversions(S)
    if S.size() < 2 then return 0
    S1, S2 ← divide(S)
    return countInversions(S1) + countInversion(S2) + mergeCount(S, S1, S2)
```

Q3 cont

```
mergeCount(S, S1, S2)
    int invCount ← 0
    while not(S1.isEmpty() or S2.isEmpty()) do
        if S1.first().key() < S2.first().key() then
            S.insertLast(S1.removeFirst())
        else
            S.insertLast(S2.removeFirst())
            invCount += (mid(S) - currentIndex(S1))      // mid(S) is index of the mid point of S
    end
    while not (S1.isEmpty()) do
        S.insertLast(S1.removeFirst())
    end
    while not (S2.isEmpty()) do
        S.insertLast(S2.removeFirst())
    end
    return invCount
```

Q4

$$T(n) = \begin{cases} 1 & , n = 1 \\ 4T\left(\frac{n}{2}\right) + n\log n & , \text{if } n \geq 2 \end{cases}$$

let $n = 2^k$

$$T(2^k) = 4T(2^{k-1}) + 2^k \log 2^k$$
$$= 4\left(4T(2^{k-2}) + 2^{k-1}\log 2^{k-1}\right) + 2^k \log 2^k$$
$$= 4^2 T(2^{k-2}) + 2^{k+1}\log 2^{k-1} + 2^k \log 2^k$$