Swapnil Daxini (V00861672) 24/02/2017

CSC 225 Assignment 2

Q1) Size of array = N

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

$\therefore$ 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. $c = 3$

$$\left.\begin{aligned} \text{1st push} &= 3(1) + 0 + 1 \\ &= 4 \\ \text{2nd push} &= 1 \\ \text{3rd push} &= 1 \end{aligned}\right\} \text{phase } 1, \text{ cost} = 6$$

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

By this we can get a relation that:

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

Doing the same for Strategy 2 :

eg. :

phase 1 : 1

phase 2 : 4

phase 3 : 8 $\quad\therefore$ number of phases : $\log_2 N$

phase 4 : 16

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

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

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

Strategy 1 :

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

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

$$= O(N^2)$$

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:

$\underline{\underline{O(n+k)}}$ 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 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$

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

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

```
merge Count(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) - current index(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

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

let $n = 2^k$

$$T(2^k) = 4T(2^{k-1}) + 2^k \log 2^k$$
$$= 4(4T(2^{k-2}) + 2^{k-1}\log 2^{k-1}) + 2^k \log 2^k$$
$$= 4^2 T(2^{k-2}) + 2^{k+1}\log 2^{k-1} + 2^k \log_2 2^k$$

Q4.

$$T(n) = 1, \quad if \quad n = 1$$
$$= 4T(\frac{n}{2}) + n\log n, \quad if \quad n \geq 2$$

Let $n = 2^b$:

$$T(2^b) = 1, \quad if \quad n = 1$$
$$= 4T(2^{b-1}) + 2^b b, \quad if \quad n \geq 2$$
$$= 4\{4T(2^{b-2}) + 2^{b-1}(b-1)\} + 2^b b$$
$$= 4^2 T(2^{b-2}) + 2^{b+1}(b-1) + 2^b b$$
$$= 4^3 T(2^{b-3}) + 2^{b+2}(b-2) + 2^{b+1}(b-1) + 2^b b$$

.

.

.

$$= 4^k T(2^{b-k}) + \sum_{i=0}^{k-1} 2^{b-i}(b-i)$$

T(1)=1 thus $2^{b-k} = 1 \quad when \ k = b$. Therefore:

$$= 4^b T(1) + \sum_{i=0}^{b-1} 2^{b-i}(b-i)$$
$$= (2^b)^2 + \sum_{i=0}^{b-1} 2^{b-i}(b-i)$$

It can be shown that $\sum_{i=0}^{b-1} 2^{b-i}(b-i) = 2(2^b \bullet b - 2^b + 1)$. Therefore putting this into the earlier equation and substituting n back in, we get:

$$T(n) = n^2 + 2n\log n - 2n + 1$$

Q5.

As the question requires us to sort items in the array in O(n) and there is a restriction on the input values, the answer is likely to be using radix sort. However, we need to know how many digits the largest number will be:

Let d equal the number of digits in the largest possible number, which in this case is $n^2 - 1$. The number of digits of the number will be dependent on the base we use. Thus:

$d = O(\log_b n)$ where b is our choice of base.

Therefore, the running time for this radix sort will be $O(d(n+b)) = O(\log_b n(n+b))$. If we make the b=n, then this will be equal to O(n). Therefore, to sort this sequence in O(n) time, we need implement Radix with the values written in base of n.

The method:

Convert all the numbers in the sequence into base n. Start by separating all the numbers by their LSB, and maintaining their order, continue separating them till their MSB. In the end, you will have a sorted list of the sequence in base n. Convert back to base 10 and you will have a sorted sequence.