Assignment 2

Guanshujie Fu, Xiaomin Qiu, Haina Lou, Yuhao Ge

March 2021

1 Problem 1

1.1

For mergesort that splits fist into k subtests, It is clear that the complexity of 'merge' algorithm remains $g(n)$. As for the 'mergesort' algorithm :

$$f(n) = a \cdot f(\lfloor \frac{n}{k} \rfloor) + b \cdot f(\lceil \frac{n}{k} \rceil) + g(n) + c \qquad a, b, c, \in R \tag{1}$$

Let $n = k^n$, hence:

$$f(k^n) = f(k^{n-1}) \cdot k + g(k^n) + c \qquad c \in R \tag{2}$$

$$\Rightarrow \quad h_n - k \cdot h_{n-1} = a' \cdot k^n + d' \qquad a', d' \in R \tag{3}$$

By solving this recurrence equation , we can derive:

$$(x-k) \cdot (x-k) \cdot (x-1) = (x-k)^2 \cdot (x-1) \tag{4}$$

Hence, $h_n = c_1 \cdot k^n + c_2 \cdot n \cdot k^n + c_3 \Rightarrow f(n) = c_1 \cdot n + c_2 \cdot n \cdot log(n) + c_3$
Clearly, $f(n) \in O(n \cdot log(n))$ and the complexity remains the same.

1.2

See codes in attached files.

1.3

To calculate the complexity precisely, we consider the comparison and movement times. For insertion sort, let's assume the probability to be equal for elements' positions In outer loop , not will execute n-1 times. In inner loop , the average comparison times for each element $data[i]$ is:

$$\frac{1 + 2 + \cdots + i}{i} = \frac{1+i}{2} \tag{5}$$

Thus the total number of comparison is:

$$C(n) = \frac{n^2 + n - 2}{4} = \sum_{i=1}^{n-1} \frac{1+i}{2} \tag{6}$$

As for the movement crimes for $data[i]$, we first consider average time:

$$\frac{0 + 1 + \cdots + i}{i} = \frac{1+i}{2} \tag{7}$$

And hence:

$$\sum_{i=1}^{n-1} (\frac{1+i}{2} + 2) = \frac{n^2 + 7n - 8}{4} \tag{8}$$

Thus,

$$C(n) + M(n) = \frac{n^2}{2} + 2n - 25$$

For merge sort, we have an induction equation:

$$T(n) = 2T(n/2) + O(n)$$

For merge function which used to merge two half lists, we need to copy all elements from half lists to temp then to $lst(2n)$. And make comparison of elements of two half lists. So $O(n)$ is $3n - 1$ in accurate. Let $2^k = n$ the function should be $T_n = 2T_{n-1} + 3 \cdot (2^k) - 1$ and hence:

$$T_n = (3 \log n - 1) \cdot n + 1.$$

Therefore, by calculation of these two equations, we derive that their complexity is almost the same when n is around 28.
As for the experiment analysis, we set 500 random numbers and different t value, with *clock_t* function which is used to show run time to find the best threshold t value which has least time complexity.
We records 10 data calculate average and find: t=30 time duration is 331, t=28 time duration is 322, t = 25 time duration is 343. We find there is no obvious difference for $t \in [25, 30]$ and we think the best threshold value is $t \in [25, 30]$, which satisfies our theoretical analysis.

2 Problem 2

2.1

We design an algorithm using 'mergesort' to make the complexity in $O(n \log n)$.

Algorithm 1 Check Demands

$List_b \leftarrow booking\_demands$
$number \leftarrow room\_number$
for $i < length(List_b)$ **do**
    $add\_tags(List_b[i].arrival\_date, 1)$
    $add\_tags(List_b[i].departure\_date, -1)$
end for
$MergeSort(List_b)$
$check \leftarrow 0$
for $i < length(List_b)$ **do**
    $check+ = List_b[i].tag$
    if $check > number$ **then**
        return 0
    end if
end for
return 1

2.2

See code in attached files.

3 Problem 3

Operations which need to be considered are 'popfront' and 'pushback'. Consider $stack_A$ and $stack_B$ where A is mainly used to pop in and B is mainly used to pop out. Let ptr_A and ptr_B be the pointers for $stack_A$ and $stack_B$. Queue is constructed based on A, B.

3.1 Pushback:

When an element is added onto the queue at the end , we just operate'push' to add it onto stack A. It is similar to 'append' and clearly the complexity is in $O(1)$:

Algorithm 2 Pushback append

$ptr_A++;$

$stack_A[ptr_A] \leftarrow new_element;$

3.2 Popfront:

For 'popfront', we first copy elements in $stack_A$ from top to end and push them onto $stack_B$. Since stack is FILO, in this case, top element of B is the head of Queue. By 'popping' out top element, 'popfront' is executed. Assume the length of queue to be n_0 and the cost for copy, pop to be c. Initially , $stack_B$ is empty and ptr_B is at the base , the cost will be:

$$C = c \cdot n_0 + c$$

Then for 'popfront' when stack B is not empty:

$$C = c \cdot n_0 + c - c \cdot n \qquad n <= n_0$$

According to amortised complexity , the complexity is in $\Theta(1)$.

Algorithm 3 Popfront

if $ptr_B == stack_{baseB}$ **then**

  while $ptr_A! = stack_{base}$ **do**

    $stack_B[++ptr_B] \leftarrow stack_A[ptr_A--]$

  end while

  ptr_B--

else

  ptr_B--

end if

4 Problem 4

4.1

For arbitrary x, take $[x - e_i]$ and $[x - e'_i]$ as two lists. As e_i is a permutation of e'_i, we can derive that:

$$length(x - e_i) = length(x - e'_i)$$

For $i \in [1, n]$, $x - e_i \in [x - e'_0, ..., x - e'_n]$ and there must exists:

$$x - e_m = x - e'_p \qquad m, p \in [1, n]$$

Hence, we can derive:

$$\prod_{i=1}^{n}(x - e_i) = \prod_{i=1}^{n}(x - e'_i)$$

Conversely, if $\prod_{i=1}^{n}(x - e_i) = \prod_{i=1}^{n}(x - e'_i)$ exists, then we can denotes that there exists m, p such that $x - e_m = x - e'_p$ holds for all factors in above equations. Hence, $e_m = e'_p$ holds for all elements in e_i, e'_i.

4.2

The evaluation $P(x) \mod p = 0$ holds iff $P(x) = 0$ or $P(x) = k \cdot p$. Let the possibility be β.

4.2.1 $P(x) = 0$

Clearly, as e_n, e'_n are different sequences, the most possible situation for $P(x) = 0$ to occur is when there exists only one different element in e_n, e'_n. As $x \in [0, p-1]$, there exists p different values for x and:

$$p > max[\frac{n}{\epsilon}, e_1, \cdots, e_n, \cdots, e'_n]$$

If $e_n, e'_n \leq p - 1$, then the possibility β for e_n, e'_n to have a common element in $[0, p-1]$ is:

$$\beta = \frac{1}{p} \cdot n$$

Since $p > \frac{n}{\epsilon}$, then $\beta < \frac{\epsilon}{n} \cdot n = \epsilon$.

4.2.2 $P(x) = k \cdot p$

This case occurs when e_n, e'_n have a common element e_i with $x - e_i = k \cdot p$ or e_n, e'_n have different elements e_i, e'_i with $x - e_i = k_1 \cdot p$ and $x - e'_i = k_2 \cdot p$.
For the first case where $x = e_i + k \cdot p$, as $x \in [0, p-1]$ and $e_i < p$, we can derive that:

$$k \cdot p > -1 \Rightarrow k \leq 0$$

In this case, we can find such a x satisfying $k \leq 0$ for a particular e_i with possibility:

$$\beta = \frac{1}{p} \cdot n < \epsilon$$

For the second case, the possibility must be smaller than the first case by induction. Hence, $\beta < \epsilon$.