CS225 Assignment 2 | Group 10

Exercise 1

(i)

$$
\begin{array}{l}
\text{sorted\_list} \leftarrow sort(\text{unsorted\_list}) = \\
\quad \textbf{IF } \text{sorted\_list} = undef \\
\quad \textbf{THEN LET } n = length(\text{unsorted\_list}) \\
\qquad \textbf{IN PAR} \\
\qquad\quad \textbf{IF } \text{sorted\_list}_1 = undef \ \wedge \ \ldots \ \wedge \ \text{sorted\_list}_\text{k} = undef \\
\qquad\quad \textbf{THEN LET } \text{list}_1 = \textbf{I}L_1.\, length(L_1) = \lceil \frac{n}{k} \rceil) \wedge \exists L'.\, concat(L_1, L') = \text{unsorted\_list} \\
\qquad\qquad\quad \textbf{IN} \quad \textbf{LET } list_2 = \textbf{I}L_2.\, length(L_2) = \lceil \frac{n - \sum_{i=1}^{1} |L_i|}{k-1} \rceil \wedge \exists L'.\, concat(L_1, L_2, L') = \text{unsorted\_list} \\
\qquad\qquad\qquad\quad \textbf{IN} \ldots \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{IN LET } list_k = \textbf{I}L_k.\, length(L_k) = \lceil \frac{n - \sum_{i=1}^{k-1} |L_i|}{1} \rceil \wedge \exists L'.\, concat(L_1, L_2, L') = \text{unsorted\_list} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{IN PAR } \text{sorted\_list}_1 = sort(\text{list}_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{sorted\_list}_\text{k} = sort(\text{list}_\text{k}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{ENDPAR} \\
\qquad\quad \textbf{ENDIF} \\
\qquad\quad \textbf{IF } \text{sorted\_list}_1 \neq undef \wedge \ldots \wedge \text{sorted\_list}_\text{k} \neq undef \\
\qquad\quad \textbf{THEN } \text{sorted\_list} \leftarrow merge(\text{sorted\_list}_1, \ldots, \text{sorted\_list}_\text{k}) \textbf{ ENDIF} \\
\qquad \textbf{ENDPAR} \\
\quad \textbf{ENDIF}
\end{array}
$$

$$
\begin{array}{l}
\text{merged\_list} \leftarrow merge(\text{inlist}_1, \ldots, \text{inlist}_\text{k}) \\
\textbf{IF } \text{merged\_list} = undef \textbf{ THEN} \\
\quad \textbf{LET } x_1 = head(\text{inlist}_1),\ \text{restlist}_1 = tail(\text{inlist}_1), \\
\qquad\qquad \ldots \\
\qquad\quad x_k = head(\text{inlist}_\text{k}),\ \text{restlist}_\text{k} = tail(\text{inlist}_\text{k}) \textbf{ IN PAR} \\
\qquad\quad \textbf{IF } x_1 \leq \forall x_i.\, \{i \neq 1\} \wedge \text{merged\_restlist} = undef \\
\qquad\quad \textbf{THEN } \text{merged\_restlist} \leftarrow merge(\text{restlist}_1, \text{inlist}_2, \ldots, \text{inlist}_\text{k}) \textbf{ ENDIF} \\
\qquad\quad \textbf{IF } x_2 \leq \forall x_i.\, \{i \neq 2\} \wedge \text{merged\_restlist} = undef \\
\qquad\quad \textbf{THEN } \text{merged\_restlist} \leftarrow merge(\text{inlist}_1, \text{restlist}_2, \text{inlist}_3, \ldots, \text{inlist}_\text{k}) \textbf{ ENDIF} \\
\qquad\qquad \ldots \\
\qquad\quad \textbf{IF } x_k \leq \forall x_i.\, \{i \neq k\} \wedge \text{merged\_restlist} = undef \\
\qquad\quad \textbf{THEN } \text{merged\_restlist} \leftarrow merge(\text{inlist}_1, \ldots, \text{inlist}_{\text{k}-1}, \text{restlist}_\text{k}) \textbf{ ENDIF} \\
\qquad\quad \textbf{IF } x_1 \leq \forall x_i.\, \{i \neq 1\} \wedge \text{merged\_restlist} \neq undef \\
\qquad\quad \textbf{THEN } \text{merged\_list} := concat([x_1], \text{merged\_restlist}) \textbf{ ENDIF} \\
\qquad\quad \textbf{IF } x_2 \leq \forall x_i.\, \{i \neq 2\} \wedge \text{merged\_restlist} \neq undef \\
\qquad\quad \textbf{THEN } \text{merged\_list} := concat([x_2], \text{merged\_restlist}) \textbf{ ENDIF} \\
\qquad\qquad \ldots \\
\qquad\quad \textbf{IF } x_k \leq \forall x_i.\, \{i \neq k\} \wedge \text{merged\_restlist} \neq undef \\
\qquad\quad \textbf{THEN } \text{merged\_list} := concat([x_k], \text{merged\_restlist}) \textbf{ ENDIF} \\
\qquad \textbf{ENDPAR} \\
\quad \textbf{ENDIF}
\end{array}
$$

Note: For the simplicity of expression of the algorithm, we omit the case where $n < thr$ in the original binomial mergesort algorithm.

Complexity Analysis:

First we see that for the merge algorithm, its complexity is in $\Theta(n)$. (We omit the complexity analysis when $n < thr$ since it is trivial.)

$$f(n) = f(\lfloor\frac{n}{k}\rfloor) + f(\lfloor\frac{n}{k}\rfloor) + \ldots + f(n-(k-1)*\lfloor\frac{n}{k}\rfloor) + g(n) + d < k * f(\lfloor\frac{n}{k}\rfloor) + g(n) + d$$

Replacing n by k^n this leads to a linear recurrence relation,

$$h_n - k \cdot h_{n-1} = a_2 \cdot k^n + b_2 + d$$

The characteristic polynomial is $(x-k)^2(x-1)$, which gives rise to

$$h_n = c_1 \cdot k^n + c_2 \cdot n \cdot k^n + c_3$$

and further

$$f_n = c_1 \cdot n + c_2 \cdot \log_k(n) \cdot n + c_3 \in O(n\log(n)|\varphi(n))$$

where the condition $\varphi(n)$ express that n is a power of k.

As $n\log(n)$ is smooth and monotone increasing (almost everywhere), we can infer that $f_n \in O(n\log(n))$.

And further because of the lower complexity bound of comparison-based sorting algorithm, $f_n \in \Theta(n\log n)$.

Finally we have shown that the complexity of the generalized Mergesort has invariant complexity under k.

Q1.3

From the pseudo-code provided in the class, we can get that:

> ```
> For selection_sort, if the length of the list is n, then we need to go
> through the loop for n times. Thus, we need:
> n steps for checking if the list is empty
> On average, searching through the list to find the smallest value needs 1/2
> (n+n(n+1)/2)=(n+3)n/4 steps
> n steps to add the smallest value into the new list
> n steps to delete the smallest value in the old list
> ```
>
> Also, we need 1 step to go out the loop. So in total, we need (n+3)n/4+1+3n steps.

> ```
> For Mergesort, if the length of list is n, then:
> ```
>
> For the division part, we need n-1 steps to separate the list into single element
> For merge part, since the depth of the division in the first part is logn, we need to deal with the whole list for logn times. In each time, we need to compare between different sublists for n times and add the smaller value into a new list for n times. So in total, we need 2n*logn in this part.*
> *So for Mergesort, the total number of steps is 2n*lognn+n-1.

To find the proper t, we let the steps of two sorting algorithms be equal.
i.e. 2n*logn+n-1=(n+3)n/4+1+3n
Finally, we get that n=26.54, so t=26.
For the experimental value, see the output of the program.

Exercise 2

Q2.1

Algorithm Description:

1. Input
   - K -- number of maximal available rooms in the hotel
   - bks -- A list (implemented using vector) of bookings
2. Output
   - **true** or **false** -- whether the bookings can be satisfied
3. Implementation of the algorithm
   The algorithm does three things
   1. Traverse through the vector list ***bks*** of bookings to find *the latest date of arrival\/departure* and *the earliest date of arrival\/departure*. This information prepares for step 2 by finding the range of the dates of interest
   2. use two arrays to record the number of arrivals and the number of departures respectively, which is implemented by using a k-sort-like algorithm and traverse through the vector list ***bks*** The indexes of the arrays correspond to dates.
   3. Initialize a counter to 0, then loop on the range of the dates of interest. In each loop (day), counter = counter + #arrivals - #departures. If counter > ***K***, then ***false*** will be returned, else true will be returned.
4. Justification on time complexity of the algorithm:
   This algorithm has complexity in $O(n)$.
   Proof:
   Step 1 takes $O(n)$ time since it simply traverses through the vector list ***bks*** which has size n.
   Step 2 takes $O(n)$ time because it simply traverses through the vector list ***bks*** which has size n, and declaring two arrays takes time $O(1)$.
   For step 3, the worst case will have complexity $O(n)$ because the range of the dates will be smaller than the length of the vector list of bookings.

Thus the total complexity is $O(n) + O(n) + O(n) = O(n)$

Exercise 3

Explained in a separate pdf file.

Exercise 4

Q4.1

Since for $\forall\ x\ P(x) = \prod_{i=1}^{n}\ (x - e_i)\ - \prod_{i=1}^{n}(x - e'_i) = 0$, take $x = [e_1, \ldots, e_{n_i}]$, get x must equal to one of $[e'_1, \ldots, e'_{n_i}]$, so for $e \in [e_1, \ldots, e_{n_i}]$, e is equal to one of $[e'_1, \ldots, e'_i]$, so $[e_1, \ldots, e_{n_i}]$ is a permutation of $[e'_1, \ldots, e'_i]$.

Q4.2

The polynomial $P(x) = \prod_{i=1}^{n} (x - e_i) - \prod_{i=1}^{n} (x - e'_i)$ is of degree $n - 1$, so according to the theorem, there are at most $n - 1$ zeros, so the probability is $\frac{n-1}{p}$;

And since we can find one solution of these $n - 1$ zeros by assigning that $e_i = e'_i$ except $e_n \neq e'_n$ since $[e_1, \ldots, e_{n_i}]$ is not the permutation of $[e'_1, .., e'_{n_i}]$.

In this way, we can take
$e_{max} = n - 1$

so we get
$p > Max\{\frac{n}{\varepsilon}, n - 1\}$
Thus we have
$Sup(\text{Probabiltity}) = \lim_{n\to\infty} \frac{n-1}{\frac{n}{\epsilon}} = \lim_{n\to\infty} \frac{n-1}{n}\epsilon = \epsilon$
Thus proven.