Homework Assignment 2

Due Date: January 31, 2021, 23:59

Note. Please note that this semester all assignments are group assignments. Further note that for the grading we will apply a "10%" rule, i.e. the maximum number of points for this assignments is 55, but 50 will be counted as 100%. Points that exceed 50 will be stored in a separate counter and used later for compensation of lost points in other assignments or (if not used up this way) the final exam.

EXERCISE 1.

(i) Generalise the mergesort algorithm splitting a list into k sublists instead of just 2. Show that the complexity remains the same.

(ii) Modify the mergesort algorithm adding a threshold value t such that for lists with a length at most t an elementary sorting algorithm (bubblesort, insertion sort, selection sort) is used instead of mergesort. Implement the modified algorithm (choose one of the three elementary sorting algorithms). $n \log n \quad n^2 \quad 2^n = n$

(iii) Provide a theoretical and experimental analysis to determine a good value for the threshold t, which optimises performance.

total points: 18

EXERCISE 2. Suppose you have to process n advance bookings of rooms for a hotel with k identical rooms. Bookings contain an arrival date and a departure date. You have to find out whether there are enough rooms in the hotel to satisfy the demands.

(i) Design an algorithm that solves this problem in time $O(n \log n)$.

Hint. Sort the set of all arrivals and departures and process it in sorted order. Choose the most appropriate sorting algorithm for this problem.

(ii) Implement your algorithm.

total points: 13

EXERCISE 3. Explain how to implement a FIFO queue using two stacks so that each FIFO operation takes amortised constant time.

allocate

total points: 8

EXERCISE 4. It is easy to check to check whether an algorithm produces a sorted output. It is less easy to check whether the output is also a permutation of the input. However, for integers there exists a fast and simple algorithm:

(i) Show that $[e_1, \ldots, e_{n_i}]$ is a permutation of $[e'_1, \ldots, e'_{n_i}]$ iff the polynomial

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

in the variable x is identically zero.

(ii) For any $\varepsilon > 0$ let p be a prime with $p > \max\{n/\varepsilon, e_1, \ldots, e_{n_i}, e'_1, \ldots, e'_{n_i}\}$. The idea is to evaluate the above polynomial $P(x)$ modulo p for a random value $x \in [0, p-1]$.

Show that if $[e_1, \ldots, e_{n_i}]$ is not a permutation of $[e'_1, \ldots, e'_{n_i}]$, then the result of the evaluation is zero with probability at most ε.

Hint. A non-zero polynomial of degree n has at most n zeroes.

total points: 16

Assignment 2

Exercise 1:

(i): For mergesort that splits list into k sublists:

It is clear that the complexity of 'merge' algorithm remains $g(n)$
As for the 'mergesort' algorithm:

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

Let $n = k^n$, hence:

$$f(k^n) = f(k^{n-1}) \cdot k + g(k^n) + c \quad , \quad c \in \mathbb{R}$$

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

By solving this recurrence equation, we can derive:

$$(x-k) \cdot (x-k) \cdot (x-1) = (x-k)^2 \cdot (x-1)$$

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, $f(n) \in \Theta(n \cdot \log n)$ and the complexity remains the same.

(ii): 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, it will excute $n-1$ times.

In inner loop, the average comparison times for each element data[i] is:

$$\frac{1+2+\cdots+i}{i} = \frac{1+i}{2} \quad , \text{ and hence the total number of comparison:}$$

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

as for the movement times for data[i], we first consider average time:

$$\frac{0+1+\cdots+i}{i} = \frac{1+i}{2} \quad , \text{ and hence:}$$

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

Thus, $C(n) + M(n) = \frac{n^2}{2} + 2n - 2.5$

Exercise 2:

(d):

This algorithm should denote the number of rooms needed in the date that most bookings overlap.

We convert the arrival and departure dates into comparable number and use mergesort to sort these dates. This results in $\Theta(n \cdot \log n)$ of complexity.

Then we loop through the ordered sets to find the number of bookings on the date that overlap most. This will have complexity of $\Theta(n)$.

Hence, overall complexity is in $\Theta(n \cdot \log n)$

Exercise 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.

① Pushback:

when an element is added into the queue at the end, we just operate 'push' to add it into stack A.

It is similar to 'append' and clearly the complexity is in $\Theta(1)$

```
ptr_A ++;
stack_A [ptr_A] = new_element;
```

② Popfront:

As for 'popfront', we first copy elements in stack A from top to end and push them into 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 excuted.

```
if ( ptr_B == stack_baseB ) {
    while ( ptr_A != stack_base) {
        stack_B [++ptr_B] = stack_A [ptr_A--];
    }
    ptr_B--;
} else { ptr_B--; }
```

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:

$$C = c \cdot n_0 + c - c \cdot n, \quad n <= n_0$$

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

Exercise 4:

(i):

① For arbitrary x, take $[x-e_i]$, $[x-e_i']$ as two lists

As $[e_1, \cdots e_n]$ is permutation of $[e_1', \cdots e_n']$, we can derive that:

$$\text{length}([x-e_i]) = \text{length}([x-e_i'])$$

For $i \in [1,n]$, $x-e_i \in [x-e_1', x-e_2' \cdots x-e_i']$ and we can assume:

$$x-e_m = x-e_p', \quad m, p \in [1,n]$$

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

② 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 $\prod(x-e_i)$, $\prod(x-e_i')$

Hence $e_m = e_p'$ holds for all elements in $[e_1 \cdots e_n]$, $[e_1' \cdots e_n']$

Clearly, $[e_1 \cdots e_n]$ is then a permutation of $[e_1' \cdots e_n']$.

(ii):