CS 225 Assignment

Group 18: Xu Ke, Wang Dingkun, Qian shuaicun, Hua Bingshen

March 2021

1 Assignment 2 Exercise 1.

1.1

For mergesort algorithm, we will have its complexity function f(n) as follows:

$$f(n) = \begin{cases} a_1n^2 + b_1n + c & if n \leq thr \\ \underbrace{f([n/k]) + f([n/k]) + \cdots + f([n/k])}_{k} + g(n) + d & else \end{cases}$$

Then replace n by k^n, we will have a linear recurrence equation, $h_n - kh_{n-1} = g(2^k) + d = a_2k^n + b_2 + d$.

The characteristic polynomial is $(x-k)^2(x-1)$, which gives rise to, $h_n = c_1k^n + c_2nk^n + c_3$.

Then let's focus on f(n) again, $f(n) = c_1n + c_2nlog_k^n + c_3 \in O(nlogn)$.

1.2

Please see the code attached.

1.3

Here we present an analysis of the threshold t. Suppose f(n) represents the time needed to sort n elements in our insertion sort. f(n) is defined as follows:

$$f(n) = \begin{cases} 2f([\frac{n}{2}]) + c_1n & n > t \\ c_2n^2 & n \leq t \end{cases}$$

From this, we can find that if t is small enough until the merge sort is more complex than insertion sort.

$t^2 \leq tlogn$

Therefore, as long as $t \in O(\log_2 n)$, the asymptotic complexity of the insertion sort will not get any worse.

2 Assignment 2 Exercise 2.

2.1

We only care about the days when there are check in and check out since

nothing change in other days. Firstly, sort arrival and departure dates together by mergesort algorithm, which takes $O(nlog(n))$. Then scan the dates from early to late, and calculate the rooms occupied and compare that with k. It takes $O(n)$. To implement this, we need to distinguish the arrival and departure, and here I mark them using another array. Mark arrival with +1 and departure with -1 and simply calculate the partial sum to get the number of rooms occupied after sorting. In general, this algorithm solves problem in time $O(nlog(n))$.

3 Assignment 2 Exercise 3.

Please see the part written by hand attached at the end.

4 Assignment 2 Exercise 4.

4.1

Since 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 and the two elements $\prod_{i=1}^{n}(x - e_i)$ and $\prod_{i=1}^{n}(x - e'_i)$ have same degree, thus the number of e_i must be equal to the number of e'_i.

Assume both e_i and e'_i are already sorted in increasing order for simplicity, that is to say we have $e_1 \leq e_2 \leq ... \leq e_{ni}$ for e_i and $e'_1 \leq e'_2 \leq ... \leq e'_{ni}$ for e'_i.

(1) It is clear that if $[e'_1, e'_2, ..., e'_{ni}]$ (let's call it set E) is a permutation of $[e_1, e_2, ..., e_{ni}]$ (let's call it set $E\prime$, all elements in set E and set $E\prime$ can be completely paired. Thus, if we sort both of them in increasing order, we can definitely get two same set. Thus, we must have $\prod_{i=1}^{n}(x - e_i) = \prod_{i=1}^{n}(x - e'_i)$, which can be represented as $P(x) = \prod_{i=1}^{n}(x - e_i) - \prod_{i=1}^{n}(x - e'_i) = 0$, hence proved.

(2) Assume $[e'_1, e'_2, ..., e'_{ni}]$ is not a permutation of $[e_1, e_2, ..., e_{ni}]$, then we must have at least one k ($1 \leq k \leq i$) which satisfies $e_k \neq e'_k$.

Then, since as defined, it is identical zero, which means its value will not depend on the variable x. Thus, we can set any value for x. Due to the condition about whether set E and set E' have different elements, let's discuss in detail here:

Condition 1: If set E and set E' have different elements. In this condition, there must exist a m which satisfies $e_m \in$ set E and $e_m \notin$ set E'. If we set x as e_m mentioned above, we will get $P(x) = P(e_m) = \prod_{i=1}^{n}(e_m - e_i) - \prod_{i=1}^{n}(e_m - e'_i)$ = 0 - $\prod_{i=1}^{n}(e_m - e'_i) \neq 0$, which contradicts our assumption.

Condition 2: If set E and set E' have different elements but some elements in set E occur in different times as in set E'. In this condition, let's extract all the part that can be paired namely set D, the equation can be rewritten as $P(x) = \prod_{e \in D}(x - e)\left[\prod_{i=1}^{n'}(e_m - e_i) - \prod_{i=1}^{n'}(e_m - e'_i)\right]$. It is clear that there must exist a x which satisfies $x \notin$ set E and $x \notin$ set E', of course, $x \notin$ set D. Then we will have $\prod_{e \in D}(x - e) \neq 0$, and $\prod_{i=1}^{n'}(e_m - e_i) - \prod_{i=1}^{n'}(e_m - e'_i) \neq 0$ as well, thus $P(x) \neq 0$, which contradicts our assumption.

Since contradiction happens in both conditions, thus our assumption that $[e'_1, e'_2, ..., e'_{ni}]$ is not a permutation of $[e_1, e_2, ..., e_{ni}]$ is wrong, hence proved.

Considering all above, we can conclude that $[e'_1, e'_2, ..., e'_{ni}]$ is a permutation of $[e_1, e_2, ..., e_{ni}]$ 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.

4.2

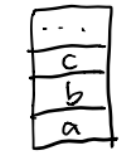
From the question above, we know that if $[e'_1, e'_2, ..., e'_{ni}]$ is not a permutation of $[e_1, e_2, ..., e_{ni}]$, $P(x) = \prod_{i=1}^{n}(x - e_i) - \prod_{i=1}^{n}(x - e'_i)$ in the variable x will not identically zero.

According to the hint: a non-zero polynomial of degree n has at most n zeroes, thus for a n degree polynomial $P(x) = \prod_{i=1}^{n}(x - e_i) - \prod_{i=1}^{n}(x - e'_i)$, the equation P(x) mod p = 0 where p is a prime number must satisfy that it has at most n roots. Since it is clear that the number of x we have is p, with x $\in$ [0,p-1], thus the probability = $\frac{thenumberofroot}{thenumberofx} = \frac{n}{p} \leq \frac{n}{n/\varepsilon} = \varepsilon$, hence proved.

Thus, if $[e'_1, e'_2, ..., e'_{ni}]$ is not a permutation of $[e_1, e_2, ..., e_{ni}]$ then the result of the evaluation is zero with probability at most ε.

Ex 3

We initialize two stacks, the in-stack and out-stack we first put elements in in-stack.



in-stack

Then we pop the elements in in-stack, and store them in out-stack.

in-stack ← pointer

out-stack ← pointer.

Now we analyze the Amortized Time

① isEmpty. We only need to decide whether the queue contains element. So the complexity is clearly in $\Theta(1)$.

② popfront.

Common case: (i) the out-stack isn't empty. then get the first element in out-stack and (ii) pop it.

if out-stack is empty. we can repeat the initialization step and (i) step.

Worst case: time complexity is $\Theta(n)$. because we need to do the initialization step. pop front element in out-stack needs $\Theta(c)$

So take potential to be $2C$. so the counter will accumulate to $2cn$. So it's enough to compensate the cost of worse case n.

So the time complexity is $\Theta(1)$

③ front, it's similar with popfront.

the amortised time complexity is $\Theta(1)$

④ push back. add a new element to the end of the queue. So we only need to put an new element on the the top of the in-stack.
so the time complexity is clearly $\Theta(1)$.

⑤ back. returns the back element of a non-empty queue without changing the queue.
If the in-stack is empty, pop all elements in out-stack and push all elements back to in-stack. Then return the top element in in-stack.
If the in-stack isn't empty. we only need to return the top element in in-stack.
The worst case time complexity is $\Theta(n)$.
Similar with the method in ② popfront.
we can get the time complexity is $\Theta(1)$.