Name: Zhiyi Guo
UNI: zg2350
Session: 002
Collaborators: Banruo Xie

Hw1—Theoretical part

1. **Solution to problem 1**

Algorithm 1

`Horner's rule`

$z = a_n$
for $i = n-1$ down to 0 **do**
    $z = zx + a_i$
end for

(a) Prove the correctness by induction

Claim: After n-times loop,

$$z = a_0 + a_1 x + \cdots + a_n x^n$$

Base: $n = 0$ and $n = 1$
When $n = 0$, the loop does not compile.

$$z = a_0$$

When $n = 1$, the loop compiles once.

$$z = a_1 x + a_0$$

Hypothesis: Assume that after n-times loop,

$$z = a_0 + a_1 x + \cdots + a_n x^n$$

Need to show the assumption is true for $n + 1$.
Induction: For now, we have the result for n-times loop ($i = n$ to $i = 1$). Only need to loop for $i = 0$.

$$z = (a_1 + a_2 x + \cdots + a_{n+1} x^n)x + a_0$$
$$\Rightarrow z = a_0 + a_1 x + \cdots + a_{n+1} x^{n+1}$$

Conclusion: The claim is proved by induction.

Running time
Inside loop, it has one addition and one multiplication. The loop runs n-times. Thus,

$$T(n) = O(n)$$

(b) Choose $z = a_i x^i$ and assume $n = 2^k$.

Algorithm 2

```
square(z, n)
1: if n=1 then
2:     return
3: end if
4: square(z^2, n/2)
```

<u>Prove the correctness by induction</u>

Claim: After k-th recursion, the algorithm ends with z^n.

Base: $k = 0$, $n = 1$. The recursion does not happen, so the algorithm ends with $z^1 = z$.

Hypothesis: Assume that after k-th recursion, the algorithm ends with z^n. Need to show the assumption is true after $k + 1$-th recursion.

Induction: For now, $k = k + 1$, $n = 2n$. Since we already have z^n after k-th recursion, we only need to recur one more to achieve $k + 1$-th.

$$(z^n)^2 = z^{2n}$$

Thus, we approve the assumption is true after $k + 1$-th recursion.

Conclusion: The claim is proved by induction.

<u>Running time</u>

Line 1 to line 2 generate $O(c)$. Thus,

$$T(n) = T(\frac{n}{2}) + O(c)$$

According to the Master Theorem ($a = 1$, $b = 2$, $k = 0$),

$$T(n) = O(\log n)$$

2. **Solution to problem 2**

Algorithm 3

```
sort(A, left, right)
if right − left ≤ 1 then
    if A[left] > A[right] then
        swap(A[left], A[right])
    end if
    return
end if
len = left + (right−left+1)/3
sort(A, left, right − len)
sort(A, left + len, right)
sort(A, left, right − len)
```

Prove the correctness by induction

Claim: $A[1,n]$ is correctly sorted after running sort$(A,1,n)$.

Base: $n=1$, $n=2$. The algorithm does not recur, it correctly sorts A in line 3.

Hypothesis: Assume sort$(A,1,i)$ correctly sorts $A[1,i]$, where $1 \leq i \leq n$. Need to show sort$(A,1,n)$ correctly sorts $A[1,n]$.

Induction: From line 8 and the hypothesis, $A[1,n-len]$ is sorted. That is,

$$A[1+len, n-len] \geq A[1, len]$$

For $A[1+len,n]$, it has at least $n-len-len-1+1 = n-2len$ elements greater or equal to $A[1,len]$. From line 9 and the hypothesis, $A[n-len,n]$ is sorted. That is,

$$A[n-len+1, n] \geq A[1+len, n-len]$$

Since $A[n-len+1,n]$ has $n-n+len-1+1 = len$ elements smaller or equal to $n-2len$. We conclude that

$$A[1,len] \leq A[1+len, n-len] \leq A[n-len+1, n]$$

Thus, $A[1,n]$ is sorted.

Conclusion: The claim is proved by induction.

Running time

Line 1 to line 4 generate $O(c)$. Thus,

$$T(n) = 3T(\frac{2}{3}n) + O(c)$$

According to the Master Theorem ($a=3$, $b=\frac{3}{2}$, $k=0$),

$$T(n) = O(n^{\log_{\frac{3}{2}} 3})$$

Since $T(n)$ is greater than the running time of mergesort ($O(n \log n)$), I will not use this algorithm for future application.

3. **Solution to problem 3**

f	g	O	o	Ω	ω	Θ
$n\log^2 n$	$6n^2\log n$	Yes	Yes	No	No	No
$\sqrt{\log n}$	$(\log\log n)^3$	No	No	Yes	Yes	No
$4\log n$	$n\log 4n$	Yes	Yes	No	No	No
$n^{3/5}$	$\sqrt{n}\log n$	No	No	Yes	Yes	No
$5\sqrt{n}+\log n$	$2\sqrt{n}$	Yes	No	Yes	No	Yes
$\frac{5^n}{n^8}$	$n^5 4^n$	No	No	Yes	Yes	No
$\sqrt{n}2^n$	$2^{n/2+\log n}$	No	No	Yes	Yes	No
$n\log 2n$	$\frac{n^2}{\log n}$	Yes	Yes	No	No	No
$n!$	2^n	No	No	Yes	Yes	No
$\log n!$	$\log n^n$	Yes	No	Yes	No	Yes

4. **Solution to problem 4**

(a) Outside the loop, it generates $O(c)$. Inside the loop, it generates $O(c)$ and the loop runs n-times. Thus,

$$T(n) = O(n) + O(c)$$

(b) The algorithm is successful when $\frac{n}{4} \leq rank \leq \frac{3n}{4}$. Thus,

$$P(successful) = \frac{\frac{3n}{4} - \frac{n}{4}}{n} = \frac{1}{2}$$

(c) In order to increase the success probability, we can recursively run the algorithm until the success rate is greater than 99%. That is, we need to find a k such that

$$1 - (1 - \frac{1}{2})^k > 99\%$$

$$\Rightarrow k > -\log(0.01)$$

$$\Rightarrow k > 6.64$$

For simplicity, we use $k = 7$ for the new algorithm.

Algorithm 4

```
median(S)
1: Uniformly at random select an item a_i
2: rank = 1
3: k = 1
4: for j = 1 to n do
5:     if a_i < a_j then
6:         rank = rank + 1
7:     end if
8: end for
9: if k > 7 then
10:     return error
11: else if n/4 ≤ rank ≤ 3n/4 then
12:     return a_i
13: else
14:     k = k + 1
15:     median(S)
16: end if
```

<u>Running time</u>

Line 1 to line 13 generate $O(n) + O(c)$ as mentioned in the question (a). Thus,

$$T(n) = T(n) + O(n) + O(c)$$

According to the Master Theorem ($a = 1$, $b = 1$, $k = 1$),

$$T(n) = O(n \log n)$$

<u>Termination</u>

Within the first 7 tries, the algorithm always terminates when a_i is correctly chosen. If all 7 tries fail, the algorithm terminates with the error, but the probability of this case is less than 1%. Thus, we conclude that the algorithm always terminates with the correct answer.

5. **Solution to problem 5**

   (a) If $|S|$ is odd, we should compile k-th_order_statistic(S, $\lceil\frac{|S|}{2}\rceil$) because the $\lceil\frac{|S|}{2}\rceil$-th smallest number is the median. If $|S|$ is even, we should compile k-th_order_statistic(S, $\frac{|S|}{2}$). Then we should compile k-th_order_statistic(S, $\frac{|S|}{2}+1$). Finally, the average of these two results is the median of S.

   (b) According to the hint, we call the split is good if it at least shrinks to $\frac{3}{4}n$. That is, a_i is a good splitter if it lies between $25th$ and $75th$ percentile of S. Thus,

$$Pr(\text{good splitter}) = \frac{1}{2}$$

Therefore, the expected number of trials to find a good splitter is 2. Let X denotes the expected number of steps on phase j. Thus,

$$E[X] = \sum_j E[X_j]$$

Since line 1 to line 5 generate cn steps. Now with at most $n(\frac{3}{4})^j$, we have

$$E[X] = \sum_j E[X_j] \leq \sum_j 2cn(\frac{3}{4})^j \leq 8cn$$

Therefore, we conclude that the expected running time is $O(n)$.