Name: Huiqian Yu
UNI: hy2635
Session: 001

Hw1—Theoretical part

1. **Solution to Problem 1**

   (a) use induction method to show that *Horner's rule* can solve this problem and store the solution in z. How many additions and multiplications does this routine use, as a function of n?

      i. **Base case:** For $n = 0$,$p(x) = a_0$,from the simple routine, we get $z = a_0$.*Horner's rule* holds for $n = 0$

      ii. **Induction Hypothesis:** For $n \geq 1$,assume that this routine correctly solve the problem for n,which means $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = z$.

      iii. **Inductive Step:** We'll show that the routine correctly solve the polynomial for $n + 1$, $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n + a_{n+1}x^{n+1}$.

      Initially $z = a_{n+1}$. We consider the first loop, $z = a_{n+1}x + a_n$. The following loops are the same as the loops for n. We have $z = a_{n+1}x^{n+1} + a_nx^n + \cdots + a_1x + a_0$ finally, and we need to prove that $z = p(x)$.
      Let $q(x) = a_1 + a_2x + a_3x^2 + \cdots + +a_nx^{n-1} + a_{n+1}x^n$, obviously, $q(x)x + a_0 = p(x)$.

      Use Induction Hypothesis, we have the solution to the polynomial $q(x)$, $w = a_1 + a_2x + a_3x^2 + \cdots + a_{n+1}x^n = q(x)$, and $wx + a_0 = q(x)x + a_0 = p(x)$.

      Moreover, $wx + a_0 = (a_1 + a_2x + a_3x^2 + \cdots + a_{n+1}x^n)x + a_0 = z = p(x)$.

      Thus, we prove that the routine holds for n+1.

   There are n loops in this routine, and each loop includes 1 addition and 1 multiplication. Thus, there are n additions and n multiplications in this routine. $T(n) = O(n)$.

   (b) Can you find a polynomial for which an alternative algorithm runs substantially faster? Consider the polynomial $p(x) = x^n$, where $n = 2^k$. Then we have use the following simple algorithm to solve this problem.

Algorithm 1 Calculate(x^{2^k})

if $k == 0$ **then**
    return x
else return (Calculate($x^{2^{k-1}}$))^2
end if

i. proof of correctness

we use induction method to prove correctness:

Base case: For $k = 0, n = 1$, $p(x) = x^1 = x$, from the simple routine, obviously we get x. The algorithm holds for $k = 0$.

Induction Hypothesis: For $k \geq 0$, assume that the algorithm correctly solve the problem for k.

Inductive Step: We have to show that for $k+1$ the result of the algorithm is equal to the solution of $q(x) = x^{2^{k+1}}$.

Consider the first step, we get result= $(\text{result1})^2$, where result_0= Calculate(x^{2^k}), according to the hypothesis, result_0 is the solution to $p(x) = x^{2^k}$. Moreover, $q(x) = x^{2^{k+1}} = p(x)^2$, so $(\text{result}_0)^2$ =result is the solution of $q(x)$. The algorithm holds for $k+1$

ii. running time

$$T(2^k) = T(2^{k-1}) + \Theta(1)$$

$$T(n) = T(n/2) + \Theta(1)$$

According to Master Theorem, $T(n) = O(\log n)$. Compared to *Horner's Rule* $T(n) = O(n)$, the new algorithm is better.

2. **Solution to Problem 2**

(a) Pseudocode

Algorithm 2 Recursive$(A, left, right)$

---

if $left == right$ **then**
    return
else if $(right - left) == 1$ **then**
    if $A[left] <= A[right]$ **then**
        return
    else SWAP$(A[left], A[right])$
    end if
else
    $flag_1 = left + [(right - left) \cdot \frac{2}{3}]$
    $flag_2 = right - [(right - left) \cdot \frac{2}{3}]$
    Recursive$(A, left, flag_1)$
    Recursive$(A, left, flag_1)$
    Recursive$(A, left, flag_1)$
end if

(b) Proof for correctness:

Suppose the algorithm sort in ascending order, and use induction method to prove correctness:

Base Case:

For $n = 0, 1, 2$,from the routine, obviously we get the sorted list.

Induction Hypothesis:

For $n \geq 0$, we assume that the recursive algorithm correctly sort the list, which means, $\forall i \leq j, A[i], A[j] \in A : A[i] \leq A[j]$.

Inductive Step:

We need to show that $\forall n+1$ $(n \geq 2)$, the algorithm works.

Consider the routine,

$$len(A, left, flag_1) \leq [\frac{2}{3}(n+1)] \leq n$$

$$len(A, flag_2, right) \leq [\frac{2}{3}(n+1)] \leq n$$

Thus, through **Recursive**$(A, left, flag_1)$ and **Recursive**$(A, flag_2, right)$, we get sorted list.

We need to prove that the result is sorted, which means: in the result,

$$\forall i \leq j,\ A[i], A[j] \in A :\ A[i] \leq A[j]$$

For $\forall i \leq j \leq flag_1\ or\ flag_1 \leq i \leq j$, it is trivial. We still need to show $\forall i \leq flag_1 \leq j$, $A[i] \leq A[j]$ holds.

By reductio ad absurdum, we assume that $\exists\ i_0 \leq flag_1 \leq j_0 \leq len(A)\ :\ A[i_0] = u > A[j_0] = v$. Before the second **Recursive**$(A, left, flag_1)$ and after **Recursive**$(A, flag_2, right)$, obviously, v is placed after $flag_1$. At the same time, u should be placed before $flag_2$, or it would contradict with **Recursive**$(A, flag_2, right)$ process, which sorted $A[flag_2 : right]$.

Before **Recursive**$(A, flag_2, right)$ and after the first **Recursive**$(A, left, flag_1)$,u is still placed before $flag_2$. At the same time v should be placed after $flag_1$, or it would contradict with the first **Recursive**$(A, left, flag_1)$ process, which sorted $A[left : flag_1]$.

Thus we have: after the first **Recursive**$(A, left, flag_1)$ process, u is set before $flag_2, v$ is set after $flag_1$. According to **Induction Hypothesis**, after the first **Recursive**$(A, left, flag_1)$ process, $\forall\ m \in [flag_2 : flag_1] \geq \forall\ n \in [left : flag_2]$, of course $\forall\ m \in A[flag_2 : flag_1] = M : m \geq u \geq v$. Then after **Recursive**$(A, flag_2, right)$ process, v should be placed after $\forall\ m \in M$, which means, v is in $A[flag_2 : flag_1]$, which contradicts what was mentioned above. Thus the assumption is invalid. Thus $\forall\ i \leq flag_1 \leq j,\ A[i] \leq A[j]$ holds.

According to the analysis of the preceding context, we've proved that in the result:

$$\forall\ i \leq j, A[i], A[j] \in A : A[i] \leq A[j].$$

Thus A is sorted by this algorithm.

(c) recurrence for its running time.

$$T(n) = 3 * T(2n/3) + \Theta(1)$$

according to master theorem:

(d) use the recurrence to bound its asymptotic running time.

According to **Master Theorem**, $T(n) = \Theta(n^{log^3_{\frac{3}{2}}}) = \omega(n^2)$

(e) No. **Insertion Sort** and **Merge Sort** are both faster than this algorithm.

3. **Solution to Problem 3**

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

4. **Solution to Problem 4**

(a) running time

There are n loops in this algorithm, each loop has constant number of primitive computational steps, and there are constant number of steps outside loops.

$$T(n) = c_1 n + c_2 = O(n)$$

(b) success probability

When the algorithm succeed happens when the random item a_i is selected exactly between the first quartile and third quartile. And the random item a_i is selected uniformly. The probability of a_i is between the first quartile and third quartile is

$$\frac{[\frac{1}{2}(n+1)] + 1}{n} \approx \frac{1}{2}$$

Thus we get success probability is $\frac{1}{2}$.

(c) improve

Consider the *Randomized Approximate Median* as a one-time probabilistic experiment that succeeds with probability $\frac{1}{2}$. We can improve the algorithm by repeat until we find a number between the first quartile and third quartile with 99% success probability after repeating at most k times.

Algorithm 3 Improved_Randomized Approx Median(S)

```
set count = 1
while count ≤ k do
    output = Randomized Approximate Median(S)
    count = count + 1
    if output ≠ error then
        return output
    end if
end while
return output
```

$$P(still\ failed\ after\ k\ times) = \frac{1}{2}^k \leq 0.01$$

then we have $k \geq 10$, thus we set $k = 10$, the improved algorithm have at least 99% success probability.

for running time:

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

5. **Solution to Problem 5**

(a) compute the median of S using this algorithm

Algorithm 4 Calculate_Median(S)

```
if length(S) is odd then
    return k − th order statistics(S, (length(S)+1)/2)
else
    return 1/2(k − th order statistics(S, length(S)/2), k − th order statistics(S, length(S)/2 + 1))
end if
end if
```

(b) The expected running time

i. Upper bound the expected time of **k-th order statistic** on a sub-problem of type j, excluding the time spent on recursive calls.

For sub-problem of type j, the new S consists of at most $n(\frac{3}{4})^j$ items, let X_j be the number of recursive calls in sub-problem. Then there are at most $X_j \cdot cn(\frac{3}{4})^j$ computational steps in sub-problem. Moreover, the sub-problem of type j is at most $[\log_{\frac{4}{3}}^{n}]th$ call in the recursive process. Then the total steps should be

$$T(n) \leq \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} X_j \cdot cn(\frac{3}{4})^j = c \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} X_j \cdot n(\frac{3}{4})^j$$

$$E[T(n)] \leq E[c \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} X_j \cdot n(\frac{3}{4})^j] = cE[\sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} X_j \cdot n(\frac{3}{4})^j] = cn \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} (\frac{3}{4})^j E[X_j]$$

and

$$E[X_j] = \sum_{i=0}^{\infty} i \cdot P(X_j = i)$$

ii. Calculate the probability that item a_i is selected such that 1/4 of the input can be thrown out (thus the input shrinks by a factor of 3/4).

This happens when a_i is between the first quartile and third quartile.

$$P(a_i \text{ results in the input shrinks by a factor of } 3/4) = \frac{1}{2}$$

iii.

$P(X_j = i)$ is the probability that, in sub-problem of type j, there are only i calls of recursive process to get the next sub-problem $j + 1$.

$$P(X_j = i) = (\frac{1}{2})^{i-1} \cdot \frac{1}{2} = (\frac{1}{2})^i$$

$$E[X_j] = \sum_{i=0}^{\infty} i \cdot P(X_j = i) = \sum_{i=0}^{\infty} i \cdot (\frac{1}{2})^i = 2$$

$$E[T] \leq c \cdot n \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} (\frac{3}{4})^j E[X_j] = 2c \cdot n \sum_{j=0}^{[\log_{\frac{4}{3}}^{n}]} (\frac{3}{4})^j \leq 2c \cdot n \sum_{j=0}^{\infty} (\frac{3}{4})^j = 8cn = O(n)$$