DIT181- Data Structure and Algorithms Assignment 1

Complexity Analysis (Big-O notation)

Question 1

a. $\sum_{i=1}^{n} i^2$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)\ (2n+1)}{6} \sim \frac{n^3}{3}$$

∴ the function above belongs to big-O complexity class $O(n^3)$

b. $\sum_{i=1}^{\log n} 2^i$

$$\sum_{i=1}^{\log n} 2^i = 2^1 + 2^2 + 2^3 + \cdots + 2^{logn} = \frac{2(1-2^{logn})}{1-2} = 2(2^{logn} - 1)$$

∴ the function above belongs to big-O complexity class $O(2^{logn})$

c. $2x^2 + 16x^3$

$$\lim_{x\to\infty} \frac{2x^2}{16x^3} = \lim_{x\to\infty} \frac{1}{8x} = 0$$

∴ the function above belongs to big-O complexity class $O\ (x^3)$

d. $(x - \log x^3)(x - 2\sqrt{x}) + 4x \log x^2$

$= x^2 - 2x^{3/2} - x * logx^3 + 2logx^3 * 2 * x^{1/2} + 4xlogx^2$
$= x^2 - 2x^{3/2} + 5\ x * logx + 12\ x^{1/2} * \log x$
Since x^2 is more dominant than $x^{3/2}$,
$xlogx$ is more dominant than $x^{1/2} * \log x$, we use L'Hospital's Rule to compare x^2 and $xlogx$

$$\lim_{x\to\infty} \frac{xlogx}{x^2} = \lim_{x\to\infty} \frac{logx}{x} = \lim_{x\to\infty} \frac{1}{x*ln2} = 0$$

∴ the term x^2 is the dominant term
∴ the function above belongs to big-O complexity class O(x^2)

Question 2

Assume that $4^n \in O(2^n)$, thus there exists constant c and k, when n >k, $4^n \leq c* 2^n$ holds. Then divide 2^n of both sides : $(\frac{4}{2})^n \leq c$,
$2^n \leq c$, when n--> ∞, there does not exist a constant c is bigger than 2^n , thus $4^n \notin O(2^n)$.

Question 3

Using Horner's method to express the polynomial
p(x)=a0xn+ a1x1+ a2x2+ ….. + anxn
by expressing it with Horner's rule, it will be
p(x) = a0 + x(a1+ x(a2+ ....+x(an-1+xan))
since it has a complexity of O(n), it reuqires n - 1 multiplications and n-1 additions
which will give an advantage of this complexity O(n2)

- Example: p(x) = 5 + 4x + 3x2+ 2x3+ x4
  Using Horner's rule
  p(x) = 5 + x(4+x(3+x(2+x)))

Question 4

Please see Question 4.java file
Using Fibonacci implementation: T(n) = T(n-1)+T(n-2)+O(1)
Using recursion for computing the sequence: T(n)= O(2n)
Then the complexity became: T(n)=O(2n-1) +O(2n-2) + O(1) = O(2n)
Solving the problem by using this code:

```
public static BI fib(int n) {
   if(n < 1) {
     return BI.zero;
   }
   if(n < 2) {
     return BI.one;
   }
  BI A = BI.zero;
  BI B = BI.one;
  BI C = A.add(b);
for (int i = 2; i < n; i++) {
A = B;
B = C;
C = B.add(A)
}
return C;
```

$T(n) = T(n-1) + T(n-2) + c$

$T(n) \leq 2 * T(n-1) + c$ (Equation 1)

$T(n-1) = T(n-2) + T(n-3) + c$

$T(n-1) \leq 2^2 * T(n-2) + c$ (Equation 2)

Substitute Equation 2 into Equation 1:

$T(n) \leq 2^2 * T(n-2) + (1+2)*c$ (Equation 3)

$T(n-2) \leq 2 * T(n-3) + c$ (Equation 4)

Substitute Equation 4 into Equation 3:

$T(n) \leq 2^3 * T(n-3) + (1+2^2)*c$

Hence, $T(n) \leq 2^1 * T(n-1) + c$ for i= 1,

$T(n) \leq 2^2 * T(n-2) + (2^0 + 2^1)*c$ for i= 2,

$T(n) \leq 2^3 * T(n-3) + (2^0 + 2^1 + 2^2)*c$ for i= 3,

…

$T(n) \leq 2^k * T(n-k) + (2^0 + 2^1 + \cdots + 2^{k-1})*c$ for i= k, (Equation 5)

$T(n-k) = T(0) = 1$, the base case n =k,

Substitute k=n into Equation 5,

$T(n) \leq 2^n + c * (2^n - 1) = (c+1) * 2^n - c$

Therefore, $T(n) = O(2^n)$

Question 5

（1） T(n) = 4*T(n/2) +$\Theta(n^2)$

$\because$ A = 4, B=2, K =2

A=B^K

$\therefore$ T(n) = $O(n^2 logn)$

（2） T(n) = 5*T(n/2) +$\Theta(n^2)$

$\because$ A=5, B=2, K =2

A>B^K

$\therefore$ T(n) = $O(n^{log5})$

Question 6

We are dividing this question into subquestions, so the problem will be divided into subproblems, the array will be divided as well.

1. left-half of the array
2. right-half of the array
3. Or we can overlaps both halves

then implement the routine of divide and conquer

```
private int maxInterval(int[] array, int start, int end) {
if ((end - start) == 1) {
if (array[end] >= array[start]) { return array[end] - array[start];
} else { return -1;
} }
int mid = (start + end) / 2;
int lowDiff = maxInterval(array, start, mid);
int highDiff = maxInterval(array, mid, end);
int i = mid;
int j = mid;
while ((i > start) && (array[i - 1] <= array[i])) {
i--; }
  while ((j < end) && (array[j] <= array[j + 1])) {
   j++;
}
int borderDiff = array[j] - array[i];
return Math.max(lowDiff, Math.max(highDiff, borderDiff));
```

}

Question 7:

Note：question is in Assignment 1.java file

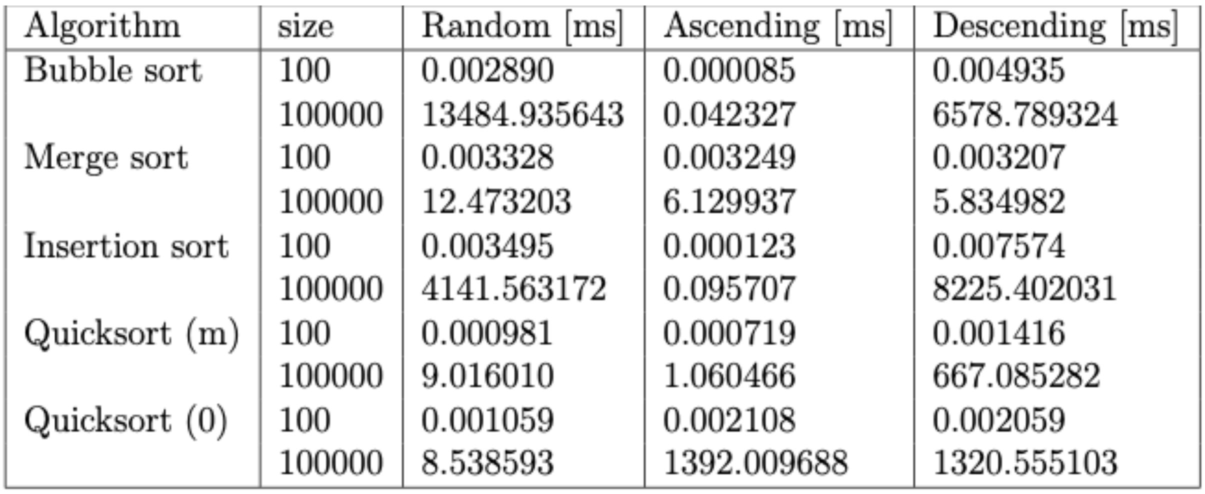
```
private static void quickSort(int[] array, int begin, int end, boolean useMedian) {
if (begin >= end) {
return;
}
int i = begin - 1;
int j = end + 1;
while (useMedian == true) {
int cen = array[(begin + end) / 2];
while (array[++i] < cen)
;
while (array[--j] > cen)
;
if (i >= j)
break;
swap(array, i, j);
}
quickSort(array, begin, i - 1, true);
quickSort(array, j + 1, end, true);

if (useMedian == false) {
int m = begin;
int n = end;
int key = array[begin];
while (m < n) {
while (key < array[n] && m < n) {
n--;
}
array[m] = array[n];
while (key >= array[m] && m < n) {
m++;
}
array[n] = array[m];
}
array[m] = key;
quickSort(array, begin, n - 1, false);
quickSort(array, n + 1, end, false);
}
}

//throw new UnsupportedOperationException();

/**
* Assignment 1 Question 7 - Mergesort
*
* @param array the array to sort
```

```
*/

public static void mergeSort(int[] array) {
Sort(array, 0, array.length - 1);
}

//throw new UnsupportedOperationException();
public static void Sort(int[] array, int low, int high) {
if (low >= high)
return;
int mid = (low + high) / 2;
if (low < high) {
Sort(array, low, mid);
Sort(array, mid + 1, high);
merge(array, low, mid, high);
}
}

public static void merge(int[] array, int low, int mid, int high) {
int[] temp = new int[high - low + 1];
int i = low;// left pointer
int j = mid + 1;// right pointer
int k = 0;

while (i <= mid && j <= high) {
if (array[i] < array[j]) {
temp[k++] = array[i++];
} else {
temp[k++] = array[j++];
}
}

while (i <= mid) {
temp[k++] = array[i++];
}

while (j <= high) {
temp[k++] = array[j++];
}

for (int k2 = 0; k2 < temp.length; k2++) {
array[k2 + low] = temp[k2];
}
}
```

Question 8:

Please see Question 8.java file

In the table there Benchmark that is executed on single thread on Kaby Lake



Algorithm	size	Random [ms]	Ascending [ms]	Descending [ms]
Bubble sort	100	0.002890	0.000085	0.004935
	100000	13484.935643	0.042327	6578.789324
Merge sort	100	0.003328	0.003249	0.003207
	100000	12.473203	6.129937	5.834982
Insertion sort	100	0.003495	0.000123	0.007574
	100000	4141.563172	0.095707	8225.402031
Quicksort (m)	100	0.000981	0.000719	0.001416
	100000	9.016010	1.060466	667.085282
Quicksort (0)	100	0.001059	0.002108	0.002059
	100000	8.538593	1392.009688	1320.555103

Here is the part of Further questions:

1. The worst that is performed of all cases is Bubble Sort
2. Comparing the the performance of the quick sort algorithm for the following two cases. We see that it is increasing in cases that have small arrays when we have a median in the sort that represents a constant overhead in quick sort and it is reducing when that becomes bigger (Ascending). When we choose an array that is already sorted and we take the first element in the array as a pivot, the sorting algorithm will end up in worst case of complexity, that won't be a problem if we choose a median routine when selecting the pivot
3. Results are the same here but when we have the array sorted from the beginning and the order is descending order.
4. If the array is small in both cases descending and ascending the insertion sort is fast, if the array becomes bigger the insertions sort becomes inefficient.
5. The performance of merge sorting is much efficient because there is so need for swapping when ascending. Considering the execution time is much better when when using merge sorting because of the distribution of the workload.