CS 206 A: 2021 SPRING FINAL EXAM

Worth 400 points

- **The total score in this exam paper is 100 points.**
- **100 points will be converted to 400 points by multiplying your earned score by 4.**

QUESTION 0. INFORMATION (WORTH 2 POINTS)

Write your name and KAIST ID number at the top of **EACH** page on your answer sheet.

QUESTION 1: 1) – 2) (12 POINTS)

For **EACH** of the functions below,

A. Give the running time in terms of ***n***, using the Big-Oh notation. (1 point)
B. Explain why it takes that running time. (5 points)

1)

```
public static void f(int n) {
          int count=0;

          if(n==0) return;
          for(int i=0;i<n;i++) {
                 count++;
          }
          f(n-1);
   }
```

2)

```
public static void f2(int n) {
          int count =0;

          for (int i = 0; i < n; i += 1) {
                 for (int j = 1; j < n; j = j * 2) {
                        count++;
                 }
          }
   }
```

QUESTION 2: 1) – 3) (26 POINTS)

Consider a dynamic array data[].

- When we push a new element, if array is full, we double the size of array data[].
- The size of data[] ***N*** is resized to precisely its actual number of elements in data[], whenever the number of elements in data[] goes less than ***N/2*** (half-full).

- Each push operation adds an element to be the last element of data[].
- Each pop operation removes the last element from data[].

1) What is the running time of returning the element at index **n/2** in data[]? Assume that data[] contains **n** elements. (worth 5 points)
   A. Give the order-of-growth in terms of **n** using the Big-Oh notation.
   B. Explain why it takes that running time.
2) What is the running time of `set(i,e)` that replaces an old element at index ***i*** with ***e***, and returns the old element in data[]? Assume that the given index ***i*** is always valid, and data[] contains **n** elements. (worth 5 points)
   A. Give the order-of-growth in terms of **n** using the Big-Oh notation.
   B. Explain why it takes that running time.
3) Carol claims that there is a sequence of ***n*** push and pop operations that requires O($\boldsymbol{n}^2$) time to execute for data[].
   - Push and pop operations can be used in any order.
   - Assume that data[] has some elements.
   - You can define how many elements data[] already contains.

   A. Is Carol's claim 'true' or 'false'? (In your answer, clearly write the full word. Otherwise, your answer will not be graded.) (worth 1 point)
   B. Justify why. (worth 15 points)

QUESTION 3: 1) – 3) (17 POINTS)

Given an array **`num`**[] that contains N distinct integers in random order, implement a quadratic-time algorithm that determines whether there are three integers that sum to exactly zero. In doing so, you are required to use one of the data structures that we have studied in class (except an array).

- You can use linear extra memory space.
- You have access to **`num`**[].

1) Specify the data structure used to implement the algorithm described above. (3 points)
2) Explain the quadratic-time algorithm that performs the given task above, using the data structure that you specified in #3-(1). (7 points)
   ⌘ Do not write the code.

3) Explain why the algorithm that you explained in #3-(2) takes quadratic time for the given array **`num`**[]. (7 points)

QUESTION 4: 1) – 2) (20 POINTS)

Consider the comparison-based sorting algorithms that we have studied in class. The contents **at some intermediate steps** during one of the comparison-based sorting algorithms are shown in each table below.

- Note that each of the tables below does **NOT** show all steps of sorting processes, but steps are shown in order from top to bottom.
- We directly start sorting the array that has the following 8 keys: [7 1 8 0 8 2 9 4].

For **EACH** of the questions 1) – 2) below,

A. Specify which comparison-based sorting algorithm that we have studied matches the contents shown below in the table. (1 point)
B. Justify why **in terms of invariants.** (9 points)

1) In the table below, (0<i< j<k;   i, j, k are distinct positive integers)

Initial	7	1	8	0	8	2	9	4
i-th step	1	7	8	0	8	2	9	4
j-th step	0	1	7	8	8	2	9	4
k-th step	0	1	2	7	8	8	9	4

2) In the table below, (0<x<y<z;   x, y, z are distinct positive integers)

Initial	7	1	8	0	8	2	9	4
x-th step	0	1	8	7	8	2	9	4
y-th step	0	1	2	7	8	8	9	4
z-th step	0	1	2	4	7	8	9	8

QUESTION 5: 1) – 2) (23 POINTS)

Consider the following **`partition()`** function below that replaces the partitioning scheme that we have studied in class for quicksort implementation.

```
// rearrange a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi]
private static int partition(Comparable[] a, int lo, int hi) {
       int j = lo;
       for (int i = lo + 1; i <= hi; i++)
              if (less(a[i], a[lo])) // strictly less
                     exch(a, i, ++j);
       exch(a, lo, j);
       return j;

}

   // is v < w ?
 private static boolean less(Comparable v, Comparable w) {
       return v.compareTo(w) < 0;
    }

    // exchange a[i] and a[j]
 private static void exch(Object[] a, int i, int j) {
       Object swap = a[i];
       a[i] = a[j];
       a[j] = swap;

    }
```

1) What is the maximum number of calls to **`less()`** during **one** call to **`partition()`** above?

A. Give the order-of-growth using the Big-Oh notation in terms of the length of ***n*** of the subarray to be partitioned. (1 point)
B. Explain why. (9 points)

2) Suppose that we use the **`partition()`** function above in quicksort with random shuffling that we have studied in class.

A. How many total calls to **`less()`** would this version of quicksort make to sort an array of ***n*** equal keys?—Give the order-of-growth using the Big-Oh notation in terms of ***n***. (1 point)
B. Explain why. (12 points)