Beijing National Day School
Department of Mathematics & Computer Science

AP Computer Science A

Test 4: `Searching, Sorting, and Recursion`

English Name: _____

Pinyin Name: _____

Mr. Alwin Tareen, Spring 2019

Exam Record	
Multiple Choice	/ 16 pts
Sorting Routines	/ 10 pts
Java Programs	/ 9 pts
Total:	/ 35 pts
Grade:	

Section I: Multiple Choice (16 points)

- Decide which is the best of the choices given, and select the correct answer by placing an "**X**" in the corresponding box.

(1pt) **1.** Consider the following code segment:

1 pt

```
public int doSomething(int n)
{
    if (n == 0)
    {
        return 0;
    }
    else
    {
        return 1 + doSomething(n - 1);
    }
}
```

What will be returned when the following method call is executed: `doSomething(6)`

- ☐ 0
- ☐ 6
- ☐ 12
- ☐ 36
- ☐ 21

(1pt) **2.** Consider the following code segment:

1 pt

```
public int doSomething(int n)
{
    if (n == 0)
    {
        return 0;
    }
    else
    {
        return n + doSomething(n - 1);
    }
}
```

What will be returned when the following method call is executed: `doSomething(6)`

- ☐ 0
- ☐ 6
- ☐ 12
- ☐ 36
- ☐ 21

2 pts

(1pt) **3.** Consider the following code segment:

1 pt

```
public int whoKnows(int n)
{
    if (n == 0 || n == 1)
    {
        return 0;
    }
    else
    {
        return n + whoKnows(n - 1) + whoKnows(n - 2);
    }
}
```

What will be returned when the following method call is executed: `whoKnows(6)`

- ☐ 6
- ☐ 15
- ☐ 20
- ☐ 38
- ☐ 120

(1pt) **4.** Consider the following code segment:

1 pt

```
public int crazy(int a, int b)
{
    if (a < b)
    {
        return a;
    }
    else
    {
        return b + crazy(a - 1, b + 1);
    }
}
```

What will be returned when the following method call is executed: `crazy(4, 2)`

- ☐ 8
- ☐ 7
- ☐ 4
- ☐ 3
- ☐ 2

2 pts

(1pt) **5.** Consider the following code segment:

1 pt

```
public void surprise(int k, String nums)
{
    if (k < nums.length())
    {
        System.out.print(nums.substring(k, k+1));
        surprise(k + 1, nums);
        System.out.print(nums.substring(k, k+1));
    }
}
```

What will be returned when the following method call is executed: `surprise(0, "123456789")`

- ☐ 987654321234566789
- ☐ 123456789987654321
- ☐ 12345678987654321
- ☐ 0123456789876543210
- ☐ 123456789000000000

(1pt) **6.** Consider the following code segment:

1 pt

```
public int doesWhat(int n)
{
    if (n <= 1)
    {
        return 1;
    }
    else if (n%2 == 0)
    {
        return n - doesWhat(n - 1);
    }
    else
    {
        return n + doesWhat(n - 1);
    }
}
```

What will be returned when the following method call is executed: `doesWhat(7)`

- ☐ 0
- ☐ 1
- ☐ 4
- ☐ 5
- ☐ 8

2 pts

(1pt) **7.** Consider the following code segment:

1 pt

```
public int prod(int p)
{
    if (p == 1)
    {
        return 1;
    }
    else
    {
        return p * prod(p - 1);
    }
}
```

What will be returned when the following method call is executed: `prod(5)`

- ☐ `10`
- ☐ `100`
- ☐ `120`
- ☐ None of the above.

(1pt) **8.** Consider the following code segment:

1 pt

```
public void func(int n)
{
    if (n == 0)
    {
        System.out.print(" ");
    }
    else
    {
        System.out.print(n + " ");
        func(n - 1);
    }
}
```

What will be returned when the following method call is executed: `func(3)`

- ☐ `3 2 1`
- ☐ `3 2`
- ☐ `3 2 1 0`
- ☐ None of the above.

2 pts

(1pt) **9.** Consider the following code segment:

1 pt

```
public int mystery(int n, int k)
{
    if (k == 0 || n == k)
    {
        return 1;
    }
    else
    {
        return mystery(n - 1, k - 1) + mystery(n - 1, k);
    }
}
```

What will be returned when the following method call is executed: `mystery(8, 3)`

- ☐ `56`
- ☐ `24`
- ☐ `15`
- ☐ `36`

(1pt) **10.** Which of the following sorting algorithms uses recursion?

1 pt

I. Insertion Sort

II. Mergesort

III. Selection Sort

- ☐ I only
- ☐ II only
- ☐ I and III
- ☐ II and III

(1pt) **11.** Which of the following techniques is used by Selection Sort?

1 pt

- ☐ Insert
- ☐ Delete
- ☐ Random
- ☐ Swap

(1pt) **12.** Sequential search is also known as:

1 pt

- ☐ Binary search
- ☐ Linear search
- ☐ Random search
- ☐ Circular search

4 pts

For questions 13 and 14, consider the following code segment which implements a linear search algorithm.

```
1  public int search(int[] nums, int key)
2  {
3      for (int i = 0; i < nums.length; i++)
4      {
5          if (nums[i] == key)
6          {
7              return i;
8          }
9      }
10     return −1;
11 }
```

(1pt) **13.** What will be returned by the `search` method if the following statements are executed:

```
int[] values = {12, 81, 19, 45, 89};
search(values, 81);
```

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ -1

1 pt

(1pt) **14.** What will be returned by the `search` method if the following statements are executed:

```
int[] values = {10, 20, 45, 60, 70};
search(values, 11);
```

- ☐ 1
- ☐ 2
- ☐ -1
- ☐ 0

1 pt

2 pts

For questions 15 and 16, consider the following code segment which implements a sorting algorithm.

```
1  public static int[] sort(int[] arr)
2  {
3      int temp = 0;
4      for (int i = 0; i < arr.length; i++)
5      {
6          for (int j = 1; j < arr.length − i; j++)
7          {
8              if (arr[j − 1] > arr[j])
9              {
10                 temp = arr[j − 1];
11                 arr[j − 1] = arr[j];
12                 arr[j] = temp;
13             }
14         }
15     }
16     return arr;
17 }
```

(1pt) **15.** What will be the result of passing an array to the above `sort` method?

1 pt

- ☐ The array will be sorted in descending order.
- ☐ The array will be sorted in ascending order.
- ☐ The array will be unsorted.
- ☐ A runtime error will occur.

(1pt) **16.** Consider the case where the array {`10, 9, 2, 15, 11`} is passed to the above `sort()` method. What will this array look like, after the second iteration of the outer `for` loop has completed?

1 pt

- ☐ {`9, 10, 2, 15, 11`}
- ☐ {`9, 2, 10, 11, 15`}
- ☐ {`2, 9, 10, 11, 15`}
- ☐ {`10, 9, 11, 15, 2`}

2 pts

Section II: Sorting Routines (10 points)

- Show the contents of each of the following arrays, after each pass of the corresponding sorting routine.
- Note that each of the sorting routines places the elements in ascending order.

(4pts) **1.** Show the contents of the following array after each pass of the outer `for` loop, for the **Selection Sort** algorithm.

4 pts

(a) (2 pts)

65	42	10	20	30	85

(b) (2 pts)

10	90	15	85	46	44

4 pts

(4pts) **2.** Show the contents of the following array after each pass of the outer `for` loop, for the **Insertion Sort** algorithm.

4 pts

(a) (2 pts)

```
65      42      10      20      30      85
```

(b) (2 pts)

```
10      90      15      85      46      44
```

(2pts) **3.** Show the contents of the following array, as it is being merged together by the **Mergesort** algorithm. Use vertical lines to demonstrate the areas where the array is separated, as the algorithm proceeds.

2 pts

```
50 | 55 | 13 | 96 | 90  | 23 | 11 | 43 | 18 | 43 | 99 | 35 | 78 | 54 | 19 | 66
```

6 pts

Section III: Java Programs (9 points)

- Show all of your work. Remember that program segments are to be written in the `Java` programming language.

(9pts) **1.** An organization interviews candidates for a variety of positions. The `Candidate` class stores the position for which a candidate is applying and the score received during the candidate's interview. The declaration of the `Candidate` class is shown below.

9 pts

```
1  public class Candidate
2  {
3      /** Constructs a new Candidate object
4       */
5      public Candidate(int idNumber, String position, double interviewScore)
6      { /* implementation not shown */ }
7
8      /** @return the position for which the candidate is applying.
9       */
10     public String getPosition()
11     { /* implementation not shown */ }
12
13     /** @return the candidate's interview score.
14      */
15     public double getInterviewScore()
16     { /* implementation not shown */ }
17
18     // There may be instance variables, constructors, and methods not shown.
19 }
```

10 pts

The `CandidatePool` class maintains a list of the candidates interviewed. The declaration of the `CandidatePool` class is shown below.

```
1  public class CandidatePool
2  {
3      /** The list of all candidates.
4       */
5      private ArrayList<Candidate> pool;
6
7      /** Constructs a new CandidatePool object.
8       */
9      public CandidatePool()
10     {
11         pool = new ArrayList<Candidate>();
12     }
13
14     /** Adds candidate to the pool
15      * @param candidate the candidate to add to the pool.
16      */
17     public void addCandidate(Candidate candidate)
18     {
19         pool.add(candidate);
20     }
21
22     /** Returns a list of candidates from the pool that have the same position as position.
23      * @param position the position of candidates to return.
24      * @return a list of candidates that have the desired position.
25      */
26     public ArrayList<Candidate> getCandidatesForPosition(String position)
27     { /* to be implemented in part (a) */ }
28
29     /** Returns the candidate from the pool with the highest interview score that has the
30      * same position as position or null if position does not match the position of
31      * any candidate.
32      * @param position the position of the candidate to return.
33      * @return candidate for position with highest interview score or null.
34      */
35     public Candidate getBestCandidate(String position)
36     { /* to be implemented in part (b) */ }
37 }
```

0 pts

(a) (4 pts) The `getCandidatesForPosition` method computes and returns a list of candidates for which the position matches `position`. If there are no candidates for which the position matches `position`, the method returns an empty list.

Complete method `getCandidatesForPosition` below.

```
/** Returns a list of candidates from the pool that have the same position as position.
 * @param position the position of candidates to return.
 * @return a list of candidates that have the desired position.
 */
public ArrayList<Candidate> getCandidatesForPosition(String position)
```

−6 pts

(b) (5 pts) The `getBestCandidate` method computes and returns the candidate with the highest interview score whose position matches `position`. If there is no candidate whose position matches `position`, the method returns `null`.

Assume that `getCandidatesForPosition` works as specified, regardless of what you wrote in part (a).

Complete method `getBestCandidate` below.

```
/** Returns the candidate from the pool with the highest interview score that has the
 * same position as position or null if position does not match the position of
 * any candidate.
 * @param position the position of the candidate to return.
 * @return candidate for position with highest interview score or null.
 */
public Candidate getBestCandidate(String position)
```

5 pts