Data Structures and Algorithms
Lab 1
ENSIA 2023-2024

Exercise 1

Write a recursive function that returns the number of 1 in the binary representation of N. Use the fact that this is equal to the number of 1 in the representation of $N/2$, plus 1 if N is odd.

Exercise 2

Write a program to solve the word puzzle problem (Figure 1). The word puzzle problem is to find given words in a two-dimensional array of characters. This problem has two inputs: a two-dimensional array of characters M, and a list of words that you have to find in M. These words may be horizontal, vertical, or diagonal in any direction (for a total of eight directions).

Exercise 3

Design a class template *Collection* that stores a collection of *Objects* in an array, along with the current size of the collection. Provide the public functions *isEmpty*, *makeEmpty*, *insert*, *remove*, and *contains*. *contains(x)* returns true if an object that is equal to x is present in the collection.
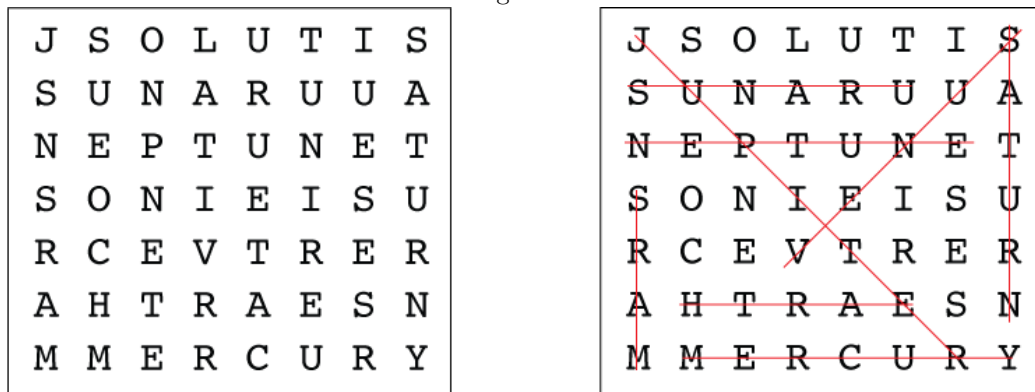
Exercise 4

Design a class template *OrderedCollection* that stores a collection of *Comparables* in an array, along with the current size of the collection. Provide the public functions *isEmpty*, *makeEmpty*, *insert*, *remove*, *findMin*, and *findMax*. The functions *findMin* and *findMax* respectively return references to the smallest and largest *comparable* in the collection. Explain what can be done if these operations are performed on an empty collection.

Exercise 5

Write a program to solve the selection problem. The selection problem determines the largest element from a set of N numbers. It can be solved using the following steps:

1. Initialize an array of size N with elements

```
J S O L U T I S
S U N A R U U A
N E P T U N E T
S O N I E I S U
R C E V T R E R
A H T R A E S N
M M E R C U R Y
```

```
J S O L U T I S
S U N A R U U A
N E P T U N E T
S O N I E I S U
R C E V T R E R
A H T R A E S N
M M E R C U R Y
```

```
EARTH        NEPTUNE
JUPITER      SATURN
MARS         URANUS
MERCURY      VENUS
```

2. Apply bubble sort in descending order to the elements

3. Return the Kth element from the array

Let $k = N/2$. Draw a table showing the running time of your program for various values of N.

Exercise 6

For each of the six program fragments in Figure 2:

1. Give an analysis of the running time (Big-O)

2. Implement the code and give the running time for several values of N

3. Compare your analysis with the actual running times

Exercise 7

1. Write a program to determine if a positive integer N is prime.

2. In terms of N, what is the worst-case running time of your program? (you should do this in $O(\sqrt{N})$)

3. Let B equal the number of bits in the binary representation of N. What is the value of B?

4. In terms of B, what is the worst-case running time of your program?

```
(1)  sum = 0;                        (4)  sum = 0;
     for( i = 0; i < n; ++i )             for( i = 0; i < n; ++i )
         ++sum;                               for( j = 0; j < i; ++j )
(2)  sum = 0;                                     ++sum;
     for( i = 0; i < n; ++i )        (5)  sum = 0;
         for( j = 0; j < n; ++j )         for( i = 0; i < n; ++i )
             ++sum;                           for( j = 0; j < i * i; ++j )
(3)  sum = 0;                                     for( k = 0; k < j; ++k )
     for( i = 0; i < n; ++i )                         ++sum;
         for( j = 0; j < n * n; ++j )  (6)  sum = 0;
             ++sum;                        for( i = 1; i < n; ++i )
                                              for( j = 1; j < i * i; ++j )
                                                  if( j % i == 0 )
                                                      for( k = 0; k < j; ++k )
                                                          ++sum;
```

5. Compare the running times to determine if a 20-bit number and a 40-bit number are prime.

6. Is it more reasonable to give the running time in terms of N or B? Why?

Exercise 8

Given an array A of integers with $A1 < A2 < A3 < \ldots < AN$, write an algorithm to determine if there is in the array an integer i such that $Ai = i$. What is the running time of your algorithm?

Exercise 9

Implement the binary search so that only one two-way comparison is performed in each iteration.

Exercise 10

Suppose you need to generate a random permutation of the first N integers. For example, {4, 3, 1, 5, 2} and {3, 1, 4, 2, 5} are legal permutations, but {5, 4, 1, 2, 1} is not because one number (1) is duplicated and another (3) is missing. This routine is often used in simulation of algorithms.

We assume the existence of a random number generator r with a method $randInt(i, j)$ that generates integers between i and j with equal probability. Here are three different algorithms:

1. Fill the array A from $A[0]$ to $A[N-1]$ as follows: To fill $A[i]$, generate random numbers until you get one that is not already in $A[0], A[1], \ldots, A[i-1]$.

2. Same as the first algorithm, but keep an extra array called *used*. When a random number *ran* is first put in the array A, set $used[ran] = true$. This means that when filling $A[i]$ with a

random number, you can see in one step whether the random number has been used, instead of the possibly i steps in the first algorithm.

3. Fill the array such that $A[i] = i + 1$. Then for$(i = 1; i < n; ++i)$ swap$(A[i], A[randInt(0, i)])$;

Now, answer the following questions:

- Prove that the three algorithms generate only legal permutations and that all permutations are equally likely.

- Give as accurate (Big-O) analysis as you can of the expected running time of each algorithm.

- Write separate programs to execute each algorithm 10 times to get a good average. Run the first program for $N = 250$, 500, 1,000, 2,000; the second program for $N = 25.000$, 50.000, 100.000, 200.000, 400.000, 800.000; and the third program for $N = 100.000$, 200.000, 400.000, 800.000, 1.600.000, 3.200.000, 6.400.000.

- Compare your analysis with the actual running times.

- What is the worst-case running time of each algorithm?

Exercise 11

A majority element in an array A of size N is an element that appears more than $N/2$ times (thus, there is at most one). For example, the array [3, 3, 4, 2, 4, 4, 2, 4, 4] has a majority element 4, whereas the array [3, 3, 4, 2, 4, 4, 2, 4] does not. If there is no majority element, your program should indicate this. Here is a sketch of an algorithm to solve the problem:

"First, a candidate majority element is found (this is the harder part). This candidate is the only element that could possibly be the majority element. The second step determines if this candidate is actually the majority. This is just a sequential search through the array. To find a candidate in the array A, form a second array B. Then compare $A1$ and $A2$. If they are equal, add one of these to B; otherwise do nothing. Then compare $A3$ and $A4$. Again, if they are equal, add one of these to B; otherwise do nothing. Continue in this fashion until the entire array is read. Then recursively find a candidate for B; this is the candidate for A, why?".

1. How does the recursion terminate?

2. How is the case where N is odd handled?

3. What is the running time of the algorithm?

4. How can we avoid using an extra array B?

5. Write a program to compute the majority element.