|  |  |  |
| --- | --- | --- |
| Hasil carian imej untuk UTEM LOGO | FACULTY OF ENGINEERING TECHNOLOGY, UNIVERSITI TEKNIKAL MALAYSIA MELAKA | |
| DISCRETE MATHEMATICS | | |
| BEEC 3413 | SEMESTER 1 | SESI 2021/2022 |
| LAB 3: SORT & SEARCHING ALGORITHM | | |
| NAME AND MATRIX NUMBER | 1. |  |
| RAHMAN KAZI ASHIKUR | B081910450 |
|  |  |
| COURSE | BEEC | |
| DATE | 12/11/ 2021 | |
| NAME OF INSTRUCTOR | AZMAN AWANG TEH | |
| EXAMINER’S COMMENT | | VERIFICATION STAMP |
| TOTAL MARKS |

## OBJECTIVES

1. To understand several important algorithms.
2. To Translate the algorithm into programming language.

## EQUIPMENTS

1. Personal Computer.
2. R Software.
3. SYNOPSIS & THEORY
   1. ALGORITHM

**Definition 1.** An *algorithm* is a finite sequence of precise instructions for performing a com- putation or for solving a problem.

The term algorithm is a corruption of the name *al-Khowarizmi*, a mathematician of the ninth century, whose book on Hindu numerals is the basis of modern decimal notation. Originally, the word *algorism* was used for the rules for performing arithmetic using decimal notation. *Algorism* evolved into the word *algorithm* by the eighteenth century. With the growing inter- est in computing machines, the concept of an algorithm was given a more general meaning, to include all definite procedures for solving problems, not just the procedures for perform- ing arithmetic.

**Example 1.** Describe an algorithm for finding the maximum (largest) value in a finite se- quence of integers.

**Solution:**

1. Set the temporary maximum equal to the first integer in the sequence. (The temporary maximum will be the largest integer examined at any stage of the procedure.)
2. Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers in the sequence.
4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

Algorithm beside written from the sets of ordered sentences like the example above, it can also written in the form of Pseudocode. **Pseudocode** provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in

a programming language.

**Note:** Pseudocode does not follow the syntax of Java, C, C++, or any other programming lan- guage.

The Pseudocode for the example above is:

# **procedure** max(a\_1, a\_2, ..., a\_n: integers) max := a\_1

**for** i := 2 **to** n

**if** max < a\_i **then** max := a\_i

# **return** max{max is the largest element}

A good algorithm should fulfil these several properties:

* *Input*. An algorithm has input values from a specified set.
* *Output*. From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
* *Definiteness*. The steps of an algorithm must be defined precisely.
* *Correctness*. An algorithm should produce the correct output values for each set of in- put values.
* *Finiteness*. An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
* *Effectiveness*. It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
* *Generality*. The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

Two simple and widely use algorithm all over the world are *sorting* and *searching*.

* 1. SORTING

Ordering the elements of a list is a problem that occurs in many contexts. For example, to produce a telephone directory it is necessary to alphabetize the names of subscribers. Sim- ilarly, producing a directory of songs available for downloading requires that their titles be put in alphabetic order. Putting addresses in order in an e-mail mailing list can determine whether there are duplicated addresses. Creating a useful dictionary requires that words be put in alphabetical order. Similarly, generating a parts list requires that we order them ac- cording to increasing part number.

* + 1. THE BUBBLE SORT

One of the sorting methods that is known widely is **bubble sort**. This method one of the sim- plest sorting algorithms, but not one of the most efficient.

**The Bubble Sort Algorithm**

**procedure** bubblesort(a\_1, ..., a\_n : real numbers with n >= 2)

**for** i := 1 **to** n - 1

**for** j := 1 **to** n - i

# **if** a\_j > a\_{j+1} **then** interchange a\_j and a\_{j+1}

{a\_1, ..., a\_n is in increasing order}

* + 1. THE INSERTION SORT

**The Insertion Sort Algorithm**

**procedure** insertion sort(a\_1, a\_2, ..., a\_n: real numbers with n >= 2)

**for** j := 2 **to** n i := 1

**while** a\_j > a\_i i := i + 1

# m := a\_j

**for** k := 0 **to** j-i-1 a\_{j-k} := a\_{j-k-1} a\_i := m

{a\_1, ..., a\_n is in increasing order}

3.3 SEARCHING

The problem of locating an element in an ordered list occurs in many contexts. For instance, a program that checks the spelling of words searches for them in a dictionary, which is just an ordered list of words. Problems of this kind are called **searching problems**. We will discuss several algorithms for searching in this section.

* + 1. THE LINEAR SEARCH

**The Linear Search Algorithm**

**procedure** linear search(x: integer, a\_1, a\_2, ..., a\_n: distinct integers) i := 1

**while** (i <= n and x <> a\_i) i := i + 1

**if** i <= n **then** location := i

**else** location := 0

# **return** location{location is the subscript of the term that equals x, or is 0 if x is not found}

* + 1. THE BINARY SEARCH

**The Binary Search Algorithm**

**procedure** binary search ($x$: integer, a\_1, a\_2, ..., a\_n: increasing integers) i := 1 {i is left endpoint of search interval}

j := n {j is right endpoint of search interval}

**while** i < j

# m := [(i + j)/2]

**if** x > a\_m **then** i := m + 1

**else** j := m

**if** x = a\_i **then** location := i

**else** location := 0

# **return** location{location is the subscript i of the term a\_i equal to x, or 0 if x is not found}

## PROCEDURE

* 1. ALGORITHM

1. Open R Gui, and create a new script.
2. Type the following program code, of **linear search** function.

# <-function(x,A){ n<-length(A)

i<-1

while(i<=n & x!= A[i]){ i<-i+1

}

if (i<= n){loc<-i} else {loc<-0} return(loc)

}

1. Run the program and observe the output.
   * define new vector

# > A <- c(1,2,3)

* + run

# > linsearch(0,A)

* + run

# > linsearch(1,A)

* + run

# > linsearch(2,A)

* + run

# > linsearch(3,A)

* + run

# > linsearch(4,A)

1. Create new functions for **Bubble Sort** algorithm.
2. Create new functions for **Binary Search** algorithm for random data.

## **RESULT**

## **linear search function**

## Graphical user interface, application Description automatically generated

## **linear search RESULT**

## Graphical user interface, text, application, Word Description automatically generated

## **Bubble Sort algorithm**

|  |
| --- |
| **bslort<-function(A) {** **n=length(A)** **for(i in 1:(n-1)){** **for(j in 1:(n-1)){** **if (A[j]>A[j+1]){** **y=A[j]** **A[j]=A[j+1]** **A[j+1]=y** **}** **}** **}** **return(A)****}****A<-c(17,23,4,12,13,14,10)****bslort(A)** |

## **RESULT**

## Text Description automatically generated with medium confidence

## **INPUT:** Logo Description automatically generated

## **OUTPUT**

## A picture containing table Description automatically generated

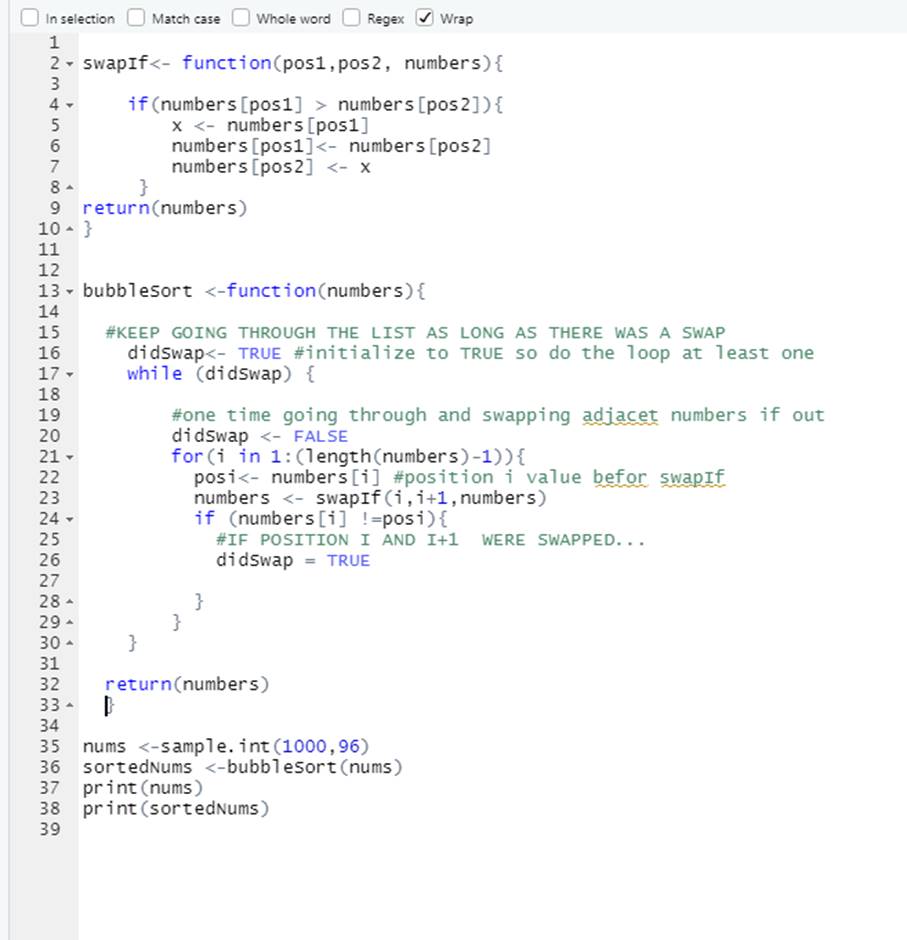
## **INPUT**

## 

## **OUTPUT**

## Table, calendar Description automatically generated

## **ANOTHER WAY TO DO BUBBLE SORT ALGORITHM.**



|  |
| --- |
| swapIf<- function(pos1,pos2, numbers){  if(numbers[pos1] > numbers[pos2]){    x <- numbers[pos1]    numbers[pos1]<- numbers[pos2]    numbers[pos2] <- x  }    return(numbers)  }  bubbleSort <-function(numbers){      #KEEP GOING THROUGH THE LIST AS LONG AS THERE WAS A SWAP      didSwap<- TRUE #initialize to TRUE so do the loop at least one      while (didSwap) {            #one time going through and swapping adjacet numbers if out          didSwap <- FALSE          for(i in 1:(length(numbers)-1)){            posi<- numbers[i] #position i value befor swapIf            numbers <- swapIf(i,i+1,numbers)            if (numbers[i] !=posi){              #IF POSITION I AND I+1  WERE SWAPPED...              didSwap = TRUE              }          }      }      return(numbers)      }    nums <-sample.int(1000,96)  sortedNums <-bubbleSort(nums)  print(nums)  print(sortedNums) |

## **Bubble Sort algorithm result**

## Text Description automatically generated with medium confidence

## Text Description automatically generated

## A picture containing table Description automatically generated

**Binary Search Algorithm**

|  |
| --- |
| **binary\_search <- function(x,A){**  **i <- 1**  **j <- length(A)**    **while(i <= j) { #i = 3 j = 5**  **m <- (j + i) / 2**  **m = as.integer(m)**  **if(x == A[m]) {**  **return(x)**  **}**    **if(x < A[m]) {**  **j <- j - 1**  **} else if(x > A[m]) {**  **i <- i + 1**  **}**  **}**  **}**  **A<-c(1,2,3,4,5)**  **#FOR OUTPUT**  **A**  **binary\_search(1,A)**  **binary\_search(2,A)**  **binary\_search(3,A)**  **binary\_search(4,A)**  **binary\_search(5,A)** |

Text

Description automatically generated

**Binary Search Algorithm RESULT**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## DISCUSSION

**linear search**

**A linear search is the simplest method of searching a data set. Starting at the beginning of the data set, each item of data is examined until a match is made. Once the item is found, the search ends. ... Find out the length of the data set.**

Example

A picture containing chart

Description automatically generated

We finding 20 we can 20 so program end at 6 index

**Bubble Sort**

A bubble sort algorithm goes through a list of data a number of times, comparing two items that are side by side to see which is out of order. It will keep going through the list of data until all the data is sorted into order. Each time the algorithm goes through the list it is called a 'pass'.

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Example: First Pass: ( 5 1 4 2 8 ) –> ( 1 5 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1. ( 1 5 4 2 8 ) –> ( 1 4 5 2 8 ), Swap since 5 > 4.

**EXAMPLE**

Diagram

Description automatically generated with medium confidence Graphical user interface, text, application, email

Description automatically generated

## CONCLUSION

Conclude what you have learned in this lab session.

**In this lab I understand 3 important algorithm which Bubble Sort Algorithm, Binary Search Algorithm, Linear Search Algorithm important algorithms. I able to Translate the algorithm into programming language. Algorithms are Important because of finite sequence of precise instructions for performing a com- pupation or for solving a problem.**

## REFERENCE

* The R Manuals, <http://www.r-project.org/>