Assignment no	1
Aim	**In second year computer engineering class, group A student's play cricket, group B students play badminton and group C students play football.** **Write a Python program using functions to compute following: - a) List of students who play both cricket and badminton b) List of students who play either cricket or badminton but not both c) Number of students who play neither cricket nor badminton d) Number of students who play cricket and football but not badminton. (Note- While realizing the group, duplicate entries should be avoided, Do not use SET built-in functions)**
Objective	To understand the concept of functions in programming languages To understand , implement SET data structure and its operations To use list or array in python to implement derived SET data structures
Outcome	To understand ,design and implement SET data structure using list or array in python To write/implement user defined functions/modules for different operations of SET in python To write menu driven, modular program in Python
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Eclipse with Python plugin or Pycharm IDE

Theory related to assignment:

In this assignment we will implements derived data structure SET data structure using list or array as primitive data structure (NOTE: we are not supposed to use inbuilt SET in python)

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Lists

- Python lists are very flexible and can hold arbitrary data.
- Lists are a part of Python's syntax, so they do not need to be declared first.
- Resize quickly
- Store heterogeneous data
- Mathematical functions can be applied directly
- List consume more memory

Arrays

- Arrays need to first be imported, or declared, from other libraries (i.e. numpy).
- Store homogenous data
- Wide range of mathematical functions can be applied directly
- Arrays are compact in size

How to use list in python:
lst=[ ]// empty list
lst.append(1) //append or add 1 element to list
print(lst) //print the list

How to use array in python:
import array as arr
a=arr.array('i',[1,2,3]) // a is array of integer

We can use list as array but we can restricts to type of elements

ADT :

Set is Objects or value definition: A finite collection with zero or more elements
Functions or operator definition:
For all S є Set and item є element
createSet() : Set //creates the empty Set
addEle(Set,item) :void //adds unique items to Set
Union(Set,Set):Set // Returns union of 2 sets
isemptySet(Set):Boolean //if Set is empty returns TRUE else returns FALSE
Intersection(Set): Set //Returns Intersection of 2 sets
End Set

ADT representation using class for Set
Class SET {
int *a; // value definition in ADT
int n; //indicate size of set
public: //Operator definition in ADT
Set create(); //creates the empty Set with n=0 and allocate memory for array a
void addelement(i,val); // appends val in Set at ith location provided val is not present in set and increment n;
void addelement(val) // append val at end and increment n
void read(); //read whole set
int read(i); // returns value at ith location from Set
bool exists(item); // if item exists in Set returns true else false
Set union(Set ,Set) // return union of 2 sets

Set intersection(Set ,Set) // return intersection of 2 sets
}

Step for implementation:

Step 1:

Program should defined six list variables which are empty initially. Declare the List for 1) students who play both cricket 2) students who play badminton 3) students who play football 3) students who play both cricket and badminton 4) students who play either cricket or badminton but not both 5) Number of students who play neither cricket nor badminton 6) Number of students who play cricket and football but not badminton

Step 2:

Accept the three sets for cricket, badminton and football by calling createSet() function which will take care of unique elements

Step 3: Define functions of union, intersection, difference/subset which accept the two list as arguments and return the answer list

Step 4: call corresponding functions with list as arguments according to menu.

Pseudo Code:

Union of two set:

procedure Set union(Set s1, Set s2)

Purpose : Finds the union of sets s1 and s2 represented as objects of Set with length of set as 'n'

Pre-condition: nothing

Post condition: Union of two sets s3 is returned as Set

1. begin
2. for i=0 to i=s1.n-1
3. begin
4. s3.addelement(s1.read(i))
5. end for
6. for i=0 to i=s2.n-1
7. begin
8. if(s3.exists(s2.read(i))==false)
9. s3.addelement(s2.read(i))
10. end if
11. end for
12. return s3

end union

Intersection of two sets:

Set intersection(Set s1, Set s2)

Purpose : Finds the union of sets s1 and s2 represented as objects of Set with length of set as 'n'

Pre-condition: nothing

Post condition: intersection of two sets s3 is returned as Set

1. begin
2. s1.read();
3. s2.read();
4. for i=0 to i=s1.n -1
5. begin
6. if(s2.exists(s1.read(i))==true) s3.addelement(s1.read(i))
7. end if
8. end for
9. return s3

end intersection

Subset of two sets:

Procedure Boolean subset (Set s1, Set s2)

Purpose : Finds the whether set2 is subset of of sets s1

Pre condition: nothing

Post condition: if set s2 is subset of set s1 then it is returned as true

1. begin
2. Subsetflag=True
3. for i=0 to s2.length-1 do
4. If s1.exists(s2[i])==0
5. Subsetflag=false
6. end for
7. return Subsetflag;

end subset;

Time Complexity: O(n)

Space Complexity: O(n)

Conclusion:

The functions for different set operations are implemented successfully using list as primitive data structure.

Review Questions:

1. What is set data structure and its applications?
2. Explain the different operations of set data structure?
3. How to use list to implement SET data structure?
4. How to use array to implement SET data structure?
5. What is list data type in python?

6. Can we add duplicate elements in SET?
7. Can we add duplicate elements in list or array?
8. SET is homogeneous or heterogeneous data type?
9. Explain the complexities of different operations of SET?

Assignment no.	2
Aim	Write a Python program to compute following operations on String: a) To display word with the longest length b) To determines the frequency of occurrence of particular character in the string c) To check whether given string is palindrome or not d) To display index of first appearance of the substring e) To count the occurrences of each word in a given string (Do not use string built-in functions)
Objective	To understand the concept of Strings in Python To apply string manipulation operations
Outcome	After executing this assignment, Students will be able to perform all the tasks without using built in functions. Students will be familiar with String module Students will be in position to use string manipulation operations
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit)

Theory related to assignment:

In this assignment We shall Learn Data type String.

- Python treats strings as contiguous series of characters delimited by single, double or even triple quotes.
- Python has a built-in string class named "str" that has many useful features.
- We can simultaneously declare and define a string by creating a variable of string type. This can be done in several ways which are as follows:

```
name = "India"

graduate = 'N'

country = name

nationality = str("Indian")
```

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

str = "Hi PICT !"

print(type(str)), then it will **print** string (str).

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str[:] = 'HELLO'

str[0:] = 'HELLO'

str[:5] = 'HELLO'

str[:3] = 'HEL'

str[0:2] = 'HE'

str[1:4] = 'ELL'

Indexing: Individual characters in a string are accessed using the subscript ([ ]) operator.

- The expression in brackets is called an index. The index specifies a member of an ordered set and in this case it specifies the character we want to access from the given set of characters in the string.
- The index of the first character is 0 and that of the last character is n-1 where n is the number of characters in the string.
- If you try to exceed the bounds (above n-1), then an error is raised.

Traversing a String: A string can be traversed by accessing character(s) from one index to another. For example, the following program uses indexing to traverse a string from first character to the last.

Example: Program to demonstrate string traversal using indexing

```
message= “Hello!”
index=0
for i in message:
        print(“message[“, index, “]=”, i)
        index+=1
```

Output:

message[0] = H

message[1] = e

message[2] = l

message[3] = l

message[4] = o

message[5] = !

a) **To display word with the longest length :**

Algorithm/Pseudocode:

Step 1: Read a Sentence

Step 2: Break up the sentence into an array of individual words (list)

Step 3: Initialize a counter variable

Step 4: Loop through each word in the array(list)

Step 5: Get the length of each word

Step 6: If the length is greater than the counter, set the counter

Step 7: Return the counter

Time Complexity: O(n), where n is the length of string.
Space: O(n), where n is the length of string.

==========

b) **To determines the frequency of occurrence of particular character in the string:**

Algorithm/ Pseudocode:

1. Start
2. Declare a string
3. Ask the user to initialize it.
4. Use a frequency array to store the frequency of each character.
5. Convert the string to a character array (list)
6. Use two for loops to calculate the frequency of each element.
7. Use the first for loop to iterate through each character of the array(list).
8. Initialize each element of the frequency array as 1(list).
9. Use another for loop to iterate through the remaining characters.
10. Check for the total occurrence of the element.
11. If the element occurs again, increment the value in the frequency array(list).
12. Set the character array to 0 to avoid counting visited characters.
13. Print the characters and their corresponding frequency.
14. Stop.

Time Complexity: O(n).
Space: O(1)

==========

c) **To check whether given string is palindrome or not :**

We are starting the algorithm by taking the string to be checked as input from the user. After that, the length of the string is calculated and stored in a variable, say 'length'. To check whether a string is palindrome or not, the given string must be reversed. To store the reversed string, we are initializing a variable 'rev' as an empty string. That being done, we are starting a loop with initial value i = length – 1. In this loop, we are reversing the string, one character at a time by performing: rev = rev + character at position i. Here, the '+' operator performs the concatenation of the characters of the string in reverse order. After that, the value of 'i' is decremented by 1. This loop runs until i >= 0. Once this loop ends, we have the reversed string in the variable rev.

We will now check whether both the strings are equal or not, ignoring the cases of the characters. If both the strings are equal, the given string is palindrome, else, the given string is not palindrome.

Algorithm/ Pseudocode:

Step 1. Start
Step 2. Read the string from the user
Step 3. Calculate the length of the string
Step 4. Initialize rev = " "[empty string]
Step 5. Initialize i = length - 1
Step 6. Repeat until i>=0:
    6.1: rev = rev + Character at position 'i' of the string
    6.2: i = i – 1
Step 7. If string = rev:
    7.1: Print "Given string is palindrome"
Step 8. Else:
    8.1: Print "Given string is not palindrome"
Step 9. Stop
Time complexity : O(n)
Space : O(1)
==========

d) To display index of first appearance of the substring

Algorithm/Pseudocode:

Step1: Read String 1

Step2: Read substring 2

Step3: Apply loop till length of string 1

Step4: Apply Loop till length of string 2

Step5: Check if ith index of str1 is equal to jth index of str 2

Step6: If true, print I and come out of the loop using Break

Step7: Print Substring not present

Time Complexity: O(n).
Space: O(1)

==========

e) To count the occurrences of each word in a given string

Algorithm/ Pseudocode:

Step1: Read String 1

Step2: Convert the string in to list

Step3: Create one more list CountList initialized with 1, of length 4

Step4: apply for loop till length of list using counter variable i

Step5: apply for loop till length of list using counter variable j

Step6: check if lst [i] is equal to list[j]

Step7: if true, increment ith element of countlist

Step8: Complete internal loop of counter variable j

Step9: Complete outer loop of counter variable i

Step10: Display the countlist

Time Complexity: O(n).
Space: O(1)

==========

Conclusion:

Students Successfully Executed the string manipulation operations without using string built in methods. Students learnt to find length of string, to calculate frequency of words in the string manually.

Review Questions:

1 What is the output of the following code ?

example = "snow world"

example[3] = 's'

print example

A. snow
B. snow world
C. Error
D. snos world

2. What is the output of “hello”+1+2+3 ?

A. hello123
B. hello
C. Error

D. hello6

3. Suppose i is 5 and j is 4, i + j is same as

A. i.__add(j)

B. i.__add__(j)

C. i.__Add(j)

D. i.__ADD(j)

4. The Index of the first character of the string is:

A. 0
B. 1
C. N-1
D. N

5. What is string in python explain String indexing and String traversing with examples.

6.Explain Concatenating, Appending and Multiplying Strings with examples.

7. Explain str () with examples.

8. String is mutable or immutable? Explain with an example.

9. Explain String formatting operator with example.

10. Explain any seven in-built string methods with example

Assignment no.	3
Aim	Write a **python** program to compute following computation on matrix: a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix
Objective	1. To understand the standard and abstract data representation methods. 2. To identify the appropriate data structure and algorithm design method for a specified application.
Outcome	1. To demonstrate a detailed understanding of behaviour of data structures like array, linked list, stack, and queue by developing programs. 2. To analyse and use effective and efficient data structures in solving various Computer Engineering domain problems. 3. To design the algorithms to solve the programming problems.
OS/Programming tools used	• 64-bit Open-source Linux or its derivative • Jupyter Notebook

Theory

Matrix is a set of elements in tabular form. The matrix can be represented using two-dimensional array data structure as shown below. An item of the matrix can be accessed using its row and column index. The basic operations which can be performed on matrix includes addition, subtraction, multiplication and transpose.

$$\text{Matrix A} = \begin{matrix} 23 & 78 & 98 \\ 04 & 21 & 33 \\ 67 & 44 & 33 \end{matrix}$$

Some of the basic constraints are to be checked before performing the operations on matrices. For example, for performing addition of two matrices, we need to have the matrices with same dimensions while multiplying two matrices the number of rows and columns of first matrix need to be same as number of columns and rows of second matrix respectively.

In python, the matrix is to be defined as a nested list. Following example shows the same.

M1 = [[8, 14, -6], [12,7,4], [-11,3,21]]

#To print the matrix

print(M1)

ADT Matrix

{A set of lists, where each list contains integers.}

Operations

1. void getMatrix(r, c): Reads a Matrix that can hold r*c elements information.
2. Matrix Transpose(A): return the matrix produced by interchanging the row and column value of every triple.
3. Matrix Add(A,B): if dimensions of a and b are the same return the matrix produced by corresponding items, namely those with identical row and column values else return error.
4. Matrix Multiply(A,B): if number of columns in a equals number of rows in B return the matrix D produced by multiplying A by B according to the formula:

   D[i][j]= Σ(A[i][k]*B[k][j])

   where D[i][j] is the $(i,j)^{th}$ element else return error.

Pseudo code:

Algorithm getMatrix()

{// to read input matrices

//M is the input matrix, r is the number of rows, c is the number of colums

1. initialse M:=[]
2. read r,c from user
3. for i:=0 to i:= r do
   - 3.1 row=[]
   - 3.2 for j:=0 to j:=c do
     - 3.2.1 read element at (i,j) position of matrix
     - 3.2.2 append element to row
4. M.append(row)

}

Algorithms

1. Matrix addition

// A and B are the matrices of dimension rows1*columns1 and rows2*columns2 respectively

// C is zero matrix of order rows1*columns1

1. Start
2. If (rows1 is not equal to rows2 or columns1 is not equal to columns2) then return ("Addition Not Possible")
3. Initialize row to zero and column to zero
4. Set C [row, column] as addition of A [row, column] and B [row, column]
5. Increment column by 1
6. Repeat step 4 and 5 till column<columns1
7. Increment row by 1
8. Repeat step 4 to 7 till row<rows1
9. Return C
10. Stop

Pseudo code

Algorithm ADD(M1,M2)

{// takes 2 matrices as input

// returns matrix M3 which is the addition of M1 and M2

1.if(M1.r==M2.r and M1.c==M2.c) *//->1 check if dimensions of input matrix are same*

{

```
    1.Initialize M3:=[]                          //->1
    2. for i:=1 to M1.r do                       //-> m+1
        2.1 row=[]                               //->m
        2.2 for j:=0 to M1.c                     //->m*(n+1)
            2.2.1 sum=M1[i][j]+M2[i][j]          //->m*n
            2.2.2 row.append(sum)                //->m*n
            2.2.3 sum:=0                         //->m*n
        2.3 M3.append(row)                       //->m
    3. return M3                                 //->1
```

}

2. Matrix subtraction

// C matrix is a zero matrix of order rows1*columns1

1. Start
2. If (rows1 is not equal to rows2 or columns1 is not equal to columns2) then return ("Subtraction Not Possible")
3. Initialize row to zero and column to zero
4. Set C [row, column] as subtraction of B [row, column] from B [row, column]
5. Increment column by 1
6. Repeat step 4 and 5 till column<columns1
7. Increment row by 1
8. Repeat step 4 to 7 till row<rows1
9. Return C
10. Stop

Pseudo code

Algorithm SUB(M1,M2)

{// takes 2 matrices as input

// returns matrix M3 which is the subtraction of M1 and M2

1.if(M1.r==M2.r and M1.c==M2.c) *//->1 check if dimensions of input matrix are same*

{

1.Initialize M3:=[] *//->1*

2. for i:=1 to M1.r do *//->m+1*

2.1 row=[] *//->m*

2.2 for j:=0 to M1.c *//->m(n+1)*

2.2.1 diff=M1[i][j]-M2[i][j] *//->mn*

2.2.2 row.append(diff) *//->mn*

2.2.3 diff:=0 *//->mn*

2.3 M3.append(row) *//->m*

3. return M3 *//->1*

}

2. else display "difference cannot be found"

}

3. Matrix multiplication

// C matrix is zero matrix of order rows1*columns2

1. Start
2. If (rows1 is not equal to columns2 or columns1 is not equal to rows2) then return ("Multiplication Not Possible")
4. Initialize row to zero and column to zero
5. Initialize i to zero
6. Set C [row, column] as C [row, column] added with multiplication of A [row, i] and B [i, column])
7. Increment i by 1
8. Repeat step 6 and 7 till i <columns1
9. Increment column by 1
10. repeat step 6 to 9 till column<columns2
11. Increment row by 1
12. repeat step 6 to 11 till row<rows1
13. return C
14. stop

Pseudo code

Algorithm MULT(M1,M2)

{// takes 2 matrices as input and returns matrix M3 which is the product of M1 and M2

1. if M1.c==M2.r do *//->1*

{

1.Initialize M3:=[] *//->1*

2. for i:=0 to M1.r do *//->m+1*

2.1 row=:[] *//->m*

2.2 for j:=0 to j:=M2.c *//->m(n+1)*

2.2.1 sum=0 *//->mn*

2.2.2 for k:=0 to k:=M2.r do *//->mn(p+1)*

2.2.2.1 sum=sum+M1[i][k]*M2[k][j] *//->mnp*

2.2.3 row.append(sum) *//->mn*

2.3. M3.append(row) *//->m*

4. return M3 *//->1*

}

2. else display "product cannot be found" *//->1*

4. Matrix transpose

// C is zero matrix of order column*rows

1. Start
2. Initialize row to zero and column to zero
4. Set C [column, row] as A [row, column]
5. Increment column by 1
6. Repeat step 4 and 5 till column<columns
7. Increment row by 1
8. repeat step 4 to 7 till row<rows
9. Return C
10. Stop

Pseudo code:

Algorithm Transpose ()

{// takes matrix as input and returns its transpose

1. input matrix *//->1*
2. for i:=0 to M1.r do *//->m+1*
    2.1 for j:=0 to M1.c do *//->m(n+!)*
        2.1.1 M3[j][i]=M1[i][j] *//->mn*
3. return M3 *//->1*

Conclusion

Two-dimensional array is the data structure identified and used for representing the matrices. Matrix operations are performed using list data structure in Python.

Review questions

1. Which data structure is to be chosen for storing matrix? Justify the same.
2. Analyse the time complexity for matrix addition and multiplication.
3. What is the space complexity for matrix subtraction operation?

Assignment no. 4	4
Aim	**a) Write a Python program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not, using Linear search and Sentinel search.** **b) Write a Python program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not, using Binary search and Fibonacci search.**
Objective	To understand the concept of searching method To find the performance of searching method To apply searching method to check whether student
Outcome	To write functions of different searching method To calculate time and space complexity of different searching method To apply functions of different searching method on student data
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit)

Theory related to assignment:

In this assignment we will implements different searching techniques; we will accept the student's data in array or list and apply the following searching operation to search an element within array or list.

The program should display the position of an element if it is found in input list or display not found message.

Linear Search:

A **linear search** or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched. A linear search runs in at worst linear time and makes at most *n* comparisons, where *n* is the length of the list. If each element is equally likely to be searched, then linear search has an average case of (n+1)/2 comparisons, but the average case can be affected if the search probabilities for each element vary.

Pseudocode Linear search:

int search(list[], int count, int key)

1. found := false
2. position := –1, index := 0
3. while (index < count and !found)
4. If (key == list[index]) then
5. found := true
6. position := index
7. break;
8. End If
9. index:=index + 1
10. end while
11. return position

end search

Time Complexity:

Best Case – O(1)
Worst Case – O(n)
Average Case – O(n)

Space Complexity: O(1)

Sentinel Search or Sentinel Linear Search:

Sentinel Linear Search as the name suggests is a type of Linear Search where the number of comparisons is reduced as compared to a traditional linear search. When a linear search is performed on an array of size N then in the worst case a total of N comparisons are made when the element to be searched is compared to all the elements of the array and (N + 1) comparisons are made for the index of the element to be compared so that the index is not out of bounds of the array which can be reduced in a Sentinel Linear Search.

Pseudocode Sentinel Search

Procedure Sentinel_Search(int arr[], int n, int x)

1. int last := arr[n - 1]; // Last element of the array
2. arr[n - 1] := x; // Element to be searched is placed at last index
3. int i = 0;
4. while (arr[i] != x)
5. i++;
6. arr[n - 1] := last; // Put the last element back
7. if ((i < n - 1) || (x == arr[n - 1]))
8. cout << x << " is present at index " << i;
9. else
10. cout << "Not found“

end Sentinel_search

	Linear Search	Sentinel Search
Best Case	**O(1) – two comparisons**	**O(1) – three comparisons**
Worst Case	**O(n) – 2N comparisons**	**O(n) – N+2 comparisons**
Average Case	**O(n) - 2N comparisons**	**O(n) – N+2 comparisons**
Space complexity	**O(1)**	**O(1)**

Binary Search

- Search a sorted array by repeatedly dividing the search interval in half.
- It begins with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- If the value of the search key is greater than the item in the middle of the interval, narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.
- We ignore half of the elements just after one comparison.
- Upper bound of Loop is decreasing by n/2 after each comparison.
- Time complexity is O ($\log_2 n$).

```
procedure binarySearch(int arr[], int s, int e, int x)
    {
    // Precondition :- arr should be sorted array
    // Postcondition :- will return position of x if present otherwise -1
1.  while (s<= e)
2.  {
3.  m :=floor((s+e) / 2);            //find mid element
4.
5.  if (arr[m] == x)                  // Check if x is present at mid
6.          return m;
7.  if (arr[m] < x)                   // If x greater, ignore left half
8.          s := m + 1;
9.  else
10.         e := m - 1;               // If x is smaller, ignore right half
11. }
12.  return -1;                 // if we reach here, then element was not present
    }
end binarySearch
```

OR

We can also implement it by using recursion.

```
Procedure binarySearch(int arr[], int s, int e, int x)
{
```

Precondition :- arr should be sorted array

Postcondition :- will return position of x if present otherwise -1

```
1. if(s<=e)
2. {
3. int m :=floor((s+e) / 2);
4. if (arr[m] == x)                    // Check if x is present at mid
5. return m;
6. if (arr[m] < x)                     // If x greater, ignore left half
7. s := m + 1;
8. return(binarySearch(arr,s,e,x));
9. else
10. e := m - 1;                        // If x is smaller, ignore right half
11. return(binarySearch(arr,s,e,x));
12. }
13. else
14. return -1;                         // if we reach here, then element was not present
15. }
```

end binarySearch

Time complexity

- Best case – O(1)
- Average, worst case - $O(\log_2 n)$

Space complexity

- Best case – O(1)
- Average, worst case - $O(\log_2 n)$

Fibonacci Search

- Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
- It is very much similar to binary search
- Works on sorted arrays
- A Divide and Conquer Algorithm
- Time complexity $\text{Log}_2 n$
- Binary search divides given array at mid but Fibonacci search divides in unequal parts
- Binary Search uses division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs.
- Fibonacci Search examines relatively closer elements in subsequent steps. So when input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.
- It is applied on nonlinear unimodal function

Procedure Fibonacci_Search(arr[ ], x, N):

1. m = 0

```
2.    while Fibo(m) < N              // repeat till mth Fibonacci no. is less than N
3.       m = m + 1
4.    offset = -1
5.    while (Fibo(m) > 1)
6.       mid = min( offset + Fibo(m - 2) , N - 1)
7.       if (x > arr[mid])
8.          m = m - 1
9.          offset = mid
10.      elif (x < arr[mid])
11.         m = m - 2
12.      else
13.         return mid
14.   end while
15.   if(!Fibo(m - 1) and arr[offset + 1] == x)
16.      return offset + 1
17.   return -1
   End Fibonacci_Search
```

Time complexity

Best case – O(1)

Average, worst case - $O(\log_2 n)$

Space complexity

Best case – O(1)

Average, worst case - $O(\log_2 n)$

Conclusion:

The functions for different searching methods are implemented successfully on student data with efficient time and space complexity.

Review Questions:

What are the various applications of linear search?

When to use linear search?

Can linear search be done on Linked Lists?

What is the best case time complexity for Linear Search?

Which search algorithm is best if data keeps changing frequently?

Can linear search be made parallel for execution on multiple CPU cores?

What is the time complexity of Linear Search for string data?

What is the time complexity of Linear Search for Integer data?

Why is the worst case time complexity same as average case for Linear Search?

Compare Binary Search vs Linear Search

Explain how does the Sentinel Search work?

Is Sentinel Linear Search better than normal Linear Search?

Explain why complexity of Binary Search is O(log n)?

What is the time complexity of fibinacci Search?