SYSTEM PROGRAMMING LAB

LAB PRACTICALS RECORD

(CSX - 326)

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)

Submitted To:	**Submitted By:**
Ms. Rupali	Aman Garg
Asst. Professor	13103050
Department of CSE	6th Semester

INDEX

S. No	Objective	Page No.	Signature
1.	To demonstrate linear and binary search.	3	
2.	To implement Sorting Techniques	6	
3.	To implement Two Pass Assembler	13	

PROGRAM -1

SEARCHING- LINEAR | BINARY

Description:

The binary search algorithm begins by comparing the target value to the value of the middle element of the sorted array. If the target value is equal to the middle element's value, then the position is returned and the search is finished. If the target value is less than the middle element's value, then the search continues on the lower half of the array; or if the target value is greater than the middle element's value, then the search continues on the upper half of the array. This process continues, eliminating half of the elements, and comparing the target value to the value of the middle element of the remaining elements - until the target value is either found (and its associated element position is returned), or until the entire array has been searched (and "not found" is returned).

Program:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

void linearSearch(vector<int> & input, const int & key){
   // Given a key and an array, it linearly  searches for the input key
   vector<int> :: iterator it = find(input.begin(), input.end(), key);
   if (it == input.end())  //we have reached end of the iterator
      cout << " Linear search couldn't locate the key: "<< key << endl;
   else
      cout << " Linear search located the key: "<< key <<" at: "<<int(it - input.begin())<<endl;
}


void binarySearch(const vector<int> & input, const int & key){
   // Given a key and a sorted array, it searches for the input key by dividing into intervals
   vector<int> newInput (input);
   sort(newInput.begin(), newInput.end());

   int l = 0, r = newInput.size() - 1, mid;
   bool foundState = false;

   while (l <= r){
      mid = (l + r)/2 ;

      if (newInput[mid] == key){
         foundState = true;
         break;
      }
```

```
        else if (newInput [mid] > key) //Key < [mid]. So, move to left interval
            r = mid - 1;
        else                //Key > [mid]. So, move to right interval
            l = mid + 1;
    }

    if (foundState == true)
        cout << " Binary search located the key: "<< key << endl;
    else
        cout << " Binary search couldn't locate the key: "<< key << endl;
}

int main(){

    // Read test data
    ifstream inf("testFile");
    if (!inf){

        fprintf(stderr,"\nError opening test file\n");
        return -1;
    }

    vector<int> input;
    int searchKey;
    char c;

    while( (c = inf.get()) != EOF)
                    input.push_back(int(c));

    inf.close();

    for (auto elem : input)
        cout << elem <<" ";

    while (true){
        cout <<"\n Enter search key: (-10 to quit) ";
        cin >> searchKey;
        if (searchKey == -10) break;

        linearSearch(input, searchKey);
        binarySearch(input, searchKey);
    }

    return 0;
}
```

```
aman@aman ~/Desktop/prog/Systems Prog/1-18 Search $ ./search
49 32 51 48 32 55 56 32 50 32 49 48 32 51 57 56 51 32 50 50 32 45 49 48 32 45 52
 56 57 32 53 54 32 56 57 32 55 56 32 55 55 32 49 50 56 10
 Enter search key: (-10 to quit) 80
 Linear search couldn't locate the key: 80
 Binary search couldn't locate the key: 80

 Enter search key: (-10 to quit) 41
 Linear search couldn't locate the key: 41
 Binary search couldn't locate the key: 41

 Enter search key: (-10 to quit) 49
 Linear search located the key: 49 at: 0
 Binary search located the key: 49

 Enter search key: (-10 to quit) 32
 Linear search located the key: 32 at: 1
 Binary search located the key: 32

 Enter search key: (-10 to quit) 71
 Linear search couldn't locate the key: 71
 Binary search couldn't locate the key: 71
```

PROGRAM - 2

SORTING ALGORITHMS

Description:

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires O(N) extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Program (MERGESORT) :

```
#include <bits/stdc++.h>
using namespace std;

void merge(vector<int> & arr, int low, int mid, int high){
    int i = low, j = mid +1, k = low;
    vector<int> c(100);

    while (i <= mid && j <= high){
        if (arr[i] < arr[j]){
            c[k] = arr[i];
            k ++;i ++;
        }
        else{
            c[k] = arr[j];
            k ++, j ++;
        }
    }
    while (i <= mid){
        c[k] = arr[i];
        k ++;i ++;
    }
    while (j <= high){
        c[k] = arr[j];
        k++, j ++;
    }
    for(i = low; i < k; i++)
        arr[i] = c[i];
```

```
    std::cout <<  endl;
    for(auto a: arr)
        cout << a <<" ";
    cout << endl;
}
void mergeSort(vector<int> &arr, int low, int high){

    if (low < high){
        int mid = (low +high) /2;
        printf("low: %d high: %d mid: %d\n",low, high, mid );

        mergeSort(arr, low, mid);
        mergeSort(arr, mid +1, high);
        merge(arr, low, mid, high);
    }
    return;
}

int main(){

    vector<int> arr {6, 5, 3, -1, 7, 10, 12, 2};
    mergeSort(arr, 0, arr.size() -1);
    return 0;
}
```

```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./mergeSort
low: 0 high: 7 mid: 3
low: 0 high: 3 mid: 1
low: 0 high: 1 mid: 0

5 6 3 -1 7 10 12 2
low: 2 high: 3 mid: 2

5 6 -1 3 7 10 12 2

-1 3 5 6 7 10 12 2
low: 4 high: 7 mid: 5
low: 4 high: 5 mid: 4

-1 3 5 6 7 10 12 2
low: 6 high: 7 mid: 6

-1 3 5 6 7 10 2 12

-1 3 5 6 2 7 10 12

-1 2 3 5 6 7 10 12
```

Program (QUICKSORT) :

```
#include <bits/stdc++.h>
using namespace std;

int partition(vector<int> &A, int low, int high){
   int pivot = A[high];
   int pivotIndex = low;

   for (int i = low; i < high; i++){
      if (A[i] <= pivot){
         swap(A[i], A[pivotIndex]);
         pivotIndex ++;
      }
   }
   swap(A[high], A[pivotIndex]);
   for (auto a : A)
   cout << a <<" ";
   cout << endl <<endl;
   return pivotIndex;
}


void quickSort(vector<int> &A, int low, int high){
   if (low < high){
      printf("quick (%d, %d)\n",low, high);
      int pivotIndex = partition(A, low, high);
      quickSort(A, low, pivotIndex -1);
      quickSort(A, pivotIndex +1, high);
   }
}


int main(){

   vector<int> A {6, 5, 3, -1, 7, 10, 12, 2};

   cout << endl;
   quickSort(A, 0, A.size() -1);
   for (auto a : A)
   cout << a <<" ";
   cout << endl;
```

```
    return 0;
}
```

```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./quickSort

quick (0, 7)
-1 2 3 6 7 10 12 5

quick (2, 7)
-1 2 3 5 7 10 12 6

quick (4, 7)
-1 2 3 5 6 10 12 7

quick (5, 7)
-1 2 3 5 6 7 12 10

quick (6, 7)
-1 2 3 5 6 7 10 12

-1 2 3 5 6 7 10 12
```

BUCKETSORT

Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. It is a distribution sort, and is a cousin of radix sortin the most to least significant digit flavour. Bucket sort is a generalization of pigeonhole sort. Bucket sort can be implemented with comparisons and therefore can also be considered a comparison sort algorithm. The computational complexity estimates involve the number of buckets. Bucket sort works as follows:

1.Set up an array of initially empty "buckets".

2.Scatter: Go over the original array, putting each object in its bucket.

3.Sort each non-empty bucket.

4.Gather: Visit the buckets in order and put all elements back into the original array.

Program (BUCKETSORT) :

```
#include <bits/stdc++.h>
using namespace std;

void bucketSort(vector<int> &A){
    int n = A.size();
    int minm = (*min_element(A.begin(), A.end()) / 10) * 10;
```

```
    int maxm = (*max_element(A.begin(), A.end()) / 10) * 10 + 10;
    int rangeM = (maxm - minm)/10;
    vector<vector<int> > buckets(rangeM);
    for(int i = 0; i < n; i++){
        int c = (A[i] / 10);
        buckets[c].push_back(A[i]);
    }

    for (int i = 0; i < buckets.size(); i++)
        sort(buckets[i].begin(), buckets[i].end());

    int index = 0;
    for (int i = 0; i < buckets.size(); i++){
        for(int j = 0; j < buckets[i].size(); j++)
            A[index++] = buckets[i][j];
    }
}

int main(){
    vector<int> A {6, 5, -1, 3, 7, 10, 12, 2};
    bucketSort(A);
    for(auto a : A)
        cout << a << " ";
    cout << endl;
    return 0;
}
```

```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./bucketSort

Total Buckets are: 2

6 goes into bucket: 0

5 goes into bucket: 0

-1 goes into bucket: 0

3 goes into bucket: 0

7 goes into bucket: 0

10 goes into bucket: 1

12 goes into bucket: 1

2 goes into bucket: 0
-1 2 3 5 6 7 10 12
```

HEAPSORT

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum. Although somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favorable worst-case O(n log n) runtime. Heapsort is an in-place algorithm, but it is not a stable sort.

Program (HEAPSORT) :

```
#include <bits/stdc++.h>
using namespace std;

int temp;
int left(int i){
    // get left child 2i + 1
     return 2*i +1;
}
int right(int i){
    // get right child 2*i +2
     return 2*i + 2;
}
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void maxHeapify(vector<int> & arr, int i, int & heapSize){
    cout <<"\n Max heapify called for index: "<<i<<" and HS: "<<heapSize<<endl;
    for (auto a : arr)
        cout << a << " " ;
    cout << endl;
    int l = left(i), r = right(i);
    int largest = i;

    if (l < heapSize && arr[l] > arr[i])
        largest = l;
    if (r < heapSize && arr[r] > arr[largest])
        largest = r;
    if (largest == i)   //we are fine. no heapify needed
        return;
    // else swap arr[i] with arr[largest]
    swap(&arr[largest], &arr[i]);
```

```
    maxHeapify(arr, largest, heapSize); }

void buildHeap(vector<int> & arr, int & heapSize){
    int mid = (arr.size() -1) /2;
    for (int i = mid; i >= 0; i --)
        maxHeapify(arr, i, heapSize );
    cout <<"\n BUILD HEAP COMPLETED\n";
}

void heapSort(vector<int> &arr){
    int heapSize = arr.size();
    buildHeap(arr, heapSize);
    for (int i = arr.size() -1; i >= 0; i--){
        swap(&arr[0], &arr[i]);
        heapSize -= 1;
        maxHeapify(arr, 0, heapSize);
    }
}

int main(){
    vector<int> arr {6, 5, -1, 3, 7, 10, 12, 2};
    heapSort(arr);
    return 0;
}
```

```
6 5 12 3 7 10 -1 2

 Max heapify called for index: 4 and HS: 8
6 7 12 3 5 10 -1 2

 Max heapify called for index: 0 and HS: 8
6 7 12 3 5 10 -1 2

 Max heapify called for index: 2 and HS: 8
12 7 6 3 5 10 -1 2

 Max heapify called for index: 5 and HS: 8
12 7 10 3 5 6 -1 2

 BUILD HEAP COMPLETED

 Max heapify called for index: 0 and HS: 7
2 7 10 3 5 6 -1 12

 Max heapify called for index: 2 and HS: 7
10 7 2 3 5 6 -1 12

 Max heapify called for index: 5 and HS: 7
10 7 6 3 5 2 -1 12

 Max heapify called for index: 0 and HS: 6
-1 7 6 3 5 2 10 12
```

```
 Max heapify called for index: 2 and HS: 5
6 5 2 3 -1 7 10 12

 Max heapify called for index: 0 and HS: 4
-1 5 2 3 6 7 10 12

 Max heapify called for index: 1 and HS: 4
5 -1 2 3 6 7 10 12

 Max heapify called for index: 3 and HS: 4
5 3 2 -1 6 7 10 12

 Max heapify called for index: 0 and HS: 3
-1 3 2 5 6 7 10 12

 Max heapify called for index: 1 and HS: 3
3 -1 2 5 6 7 10 12

 Max heapify called for index: 0 and HS: 2
2 -1 3 5 6 7 10 12

 Max heapify called for index: 0 and HS: 1
-1 2 3 5 6 7 10 12

 Max heapify called for index: 0 and HS: 0
-1 2 3 5 6 7 10 12
```

PROGRAM – 3

TWO PASS ASSEMBLER

Description:

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like

Program (Python) :

```
from __future__ import print_function
import re

# --------------Source Assembly Files---------------
sourceCode = "sourceFile.as"
mcOpTableFile = "mcOpTable"
passOneOutput = "pass I"
passTwoOutput = "pass II"
# --------------Source Assembly Files---------------


# --------------instruction Classes---------------
imperativeInstructions = ['MOVEM', 'MOVER',
                          'ADD', 'SUB', 'MUL', 'BC', 'LTORG']

ASS_DIRECTIVES = ['START', 'END', 'ORIGIN',  'EQU', 'LTORG',
                  'PURGE', 'USING', 'SEGMENT', 'END', 'ASSUME',
                  'PUBLIC', 'EXTERN', 'BALR']

REGISTER_LIST = ['AREG', 'BREG', 'CREG', 'DREG']

declarativeInstructions = ['DS', 'DC']
# --------------instruction Classes---------------

# location Counter defaults to 0
LC = 0
patternLiteral = re.compile("='(\w)'")

# PASS 1 uses OPTAB, SYMTAB, LITTAB, POOLTAB

OPTAB = {}
# --------------Pass I O/P---------------
SYMTAB = {}
LITTAB = {}
# --------------Pass I O/P---------------
```

```
def startswithAssDirective(line):
    """Determines whether the line starts not with a label"""

    for i in ASS_DIRECTIVES + imperativeInstructions + declarativeInstructions:
        if line[0] == i:
            return True
    return False


def getAddressFromSymTab(reqdSymbol):
    """Gets the required address of the symbol from SYMTABLE"""
    try:
        return SYMTAB[reqdSymbol]
    except KeyError:
        return None


def updateSymTab(reqdSymbol, reqdAdd):
    """Updates the address of the symbol with the reqdAdd"""
    SYMTAB[reqdSymbol] = reqdAdd


def imperativeStatement(line):
    """Returns true if the line is an imperative statement"""
    for i in imperativeInstructions:
        if i in line:
            return True
    return False


def declarativeStatement(line):
    """Returns true if line is a declarative statement"""
    for i in declarativeInstructions:
        if i in line:
            return True
    return False


def getSizeFromMOT(line):
    """Given an opcode and its type, return its machine size"""
    for i in line:
        if i in OPTAB:
            return OPTAB[i][1]


def getCodeFromMOT(line):
    """Given an opcode and its type, return its machine code"""
```

```
    for i in line:
        if i in OPTAB:
            return OPTAB[i][0]

def isLiteral(s):
    """Returns true if a string contains a literal"""
    p = patternLiteral.search(s)
    if p is None:
        return (False, None)
    return (True, p.groups()[0])

def literalImmediate(line):
    """Returns true if  a line contains a literal string"""
    literalList = []
    for i in line:
        res = isLiteral(i)
        if res[0]:
            literalList.append(res[1])
    return literalList

def passOne(fileHandle):
    """Generates SYMTAB LITTAB POOLTAB given a source file"""
    global LC
    LTORG_SET = False

    # Open source file for reading
    f = file(sourceCode)
    literalPending = []

    for i in f.readlines():
        line = i.split()

        if 'END' in line:
            # process remaining literals in the literal pool
            for i in literalPending:
                LITTAB[i] = LC
                literalPending.remove(i)
                LC += 1
            break

        LC += 1
        literalPending.extend(literalImmediate(line))

        # if symbol is present in lable field
        if not startswithAssDirective(line) and len(line) != 1:
            # An only literal is not a label
            SYMTAB[line[0]] = LC
```

```
        for lit in literalPending:
            # if there's a literal in the line
            # Check entry in littable whether it has been mapped to a loc
            # if no entry, create a new entry as None.
            if lit not in LITTAB:
                LITTAB[lit] = None

        if LTORG_SET:
            # Then revisit the mapping to the current location and increase LC
            if not isLiteral(line[0]):
                LTORG_SET = False
                break
            for i in literalPending:
                LITTAB[i] = LC
                literalPending.remove(i)
                LC += 1

        if 'LTORG' in line:
            LTORG_SET = True

        # if a start statement
        elif 'START' in line:
            # Update LC to denote main program
            LC = int(line[1])
            SYMTAB[line[0]] = LC

        elif 'EQU' in line:
            # place the address of the third value as that of the first
            currAdd = getAddressFromSymTab(line[2])
            updateSymTab(line[0], currAdd)

        elif imperativeStatement(line) or declarativeStatement(line):
            size = getSizeFromMOT(line)
            LC += size

    # Close source file
    f.close()

    # ------------OUTPUT OF PASS-1 ----------------
    fileHandle.write('\nSYMBOL TABLE: ')
    for i in SYMTAB.iteritems():
        fileHandle.writelines(('\n' + str(i[0]) + '\t' + str(i[1])))

    fileHandle.write('\n\nLITERAL TABLE ')
    for i in LITTAB.iteritems():
        fileHandle.writelines(('\n' + str(i[0]) + '\t' + str(i[1])))
    # -----------OUTPUT OF PASS 2 -------------------
def passTwo(fileHandle):
```

```
    """Generates final machine code using symbol table, literal table"""
    global LC

    # Open source file for reading
    f = file(sourceCode)

    for i in f.readlines():
        line = i.split()
        if 'END' in line:
            break

        if 'START' in line:
            LC = SYMTAB['START']

        elif imperativeStatement(line) or declarativeStatement(line):
            # Process Operands carefully
            operands = []

            for possibleOp in line:
                if possibleOp in REGISTER_LIST:
                    operands.append(REGISTER_LIST.index(possibleOp) + 1)

                elif possibleOp in SYMTAB:
                    operands.append(SYMTAB[possibleOp])

                elif possibleOp in LITTAB:
                    operands.append(LITTAB[possibleOp])

            opcode = getCodeFromMOT(line)
            length = getSizeFromMOT(line)
            fileHandle.write(('\n' + str(LC) + '\t'))
            for i in operands:
                fileHandle.write(str(i) + '\t')
            fileHandle.write(('\t' + str(opcode) + '\t' + str(length)))
            LC += length

    f.close()

def main():
    # Read input source Files, mcOPtable and pseudoOptable
    fOp = file(mcOpTableFile)
    for i in fOp.readlines():
        l = i.split()
        OPTAB[l[0]] = (l[1], int(l[2]))
    fOp.close()


    pA = file(passOneOutput, 'w')
```

```
    pA.writelines('\n-----------OUTPUT OF PASS I------------\n')
    passOne(pA)
    pA.writelines('\n\n-----------OUTPUT OF PASS I-----------\n')

    # Read symbol table, literal table and machine opcode table and produce final code
    pB = file(passTwoOutput, 'w')
    pB.write('\n-----------OUTPUT OF PASS II-----------\n')
    passTwo(pB)
    pB.write('\n\n-----------OUTPUT OF PASS II-----------\n')

if __name__ == '__main__':
    main()
```

INPUT:

Source File

```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat sourceFile.as
     START 200
     MOVER AREG ='5'
     MOVEM AREG A
LOOP MOVER AREG A
     MOVER CREG B
     ADD CREG ='1'
     BC ANY NEXT
     LTORG
        ='5'
        ='1'
NEXT SUB AREG ='4'
     BC LT BACK
LAST STOP
     MUL CREG B
A    DS 1
BACK EQU LOOP
B    DS 1
     END
```

Machine Opcode Table

```
aman@aman ~/Desktop/prog/Systems Prog/
2-15 Assembler $ cat mcOpTable
START R#3 1
ADD 3E 2
MOVEM 14 2
BC 5D 3
DC R#9 1
MUL 1F 2
MOVER 04 1
LTORG R#8 1
DS R#7 1
SUB 4D 2
```

OUTPUTS:

Pass I Output File

```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat pass\ I

-----------OUTPUT OF PASS I------------

SYMBOL TABLE:
A       236
B       239
LAST    232
BACK    206
NEXT    223
START   200
LOOP    206

LITERAL TABLE
1       223
5       219
4       227

-----------OUTPUT OF PASS I-----------
```

Pass II Output File

```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat pass\ II

-----------OUTPUT OF PASS II-----------

LC      MNEMONIC    OPERANDS        LENGTH
200     1               04              1
201     1       236             14      2
203     206     236             04      1
204     3       239             04      1
205     3               3E              2
207     223             5D              3
210             R#8                     1
211     223     1               4D      2
213     206             5D              3
216     3       239             1F      2
218     236     223             R#7     1
219     239     223             R#7     1

-----------OUTPUT OF PASS II-----------
```