SEARCHING & SORTING

```
/*
* ***********************
* SEARCHING AND SORTING *
* ***********************
*/

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#define MAX 20

int * a, n;
void inputArray() {
 int i;
 if(a != NULL) {
     free(a);
 }
 printf("ENTER SIZE OF ARRAY :");
 scanf("%d",&n);
 a = (int *) malloc (sizeof(int) * n);
 for(i = 0 ; i < n ; i++) {
     printf("ENTER ELEMENT %d : ",i);
     scanf("%d",&a[i]);
     }
}

void displayArray()  {
 int i;
     printf("\nARRAY : ");
 for(i = 0 ; i < n ; i++) {
          printf("%d ",a[i]);
      }
 }

int bSearch(int key, int * a, int lb, int ub) {
     int mid = (lb + ub) / 2;
     if(lb <= ub) {
          if(key == a[mid]) {
           return mid;
          }
```

```
          if(key < a[mid]) {
           return bSearch(key, a, 0, mid-1);
          }
          if(key > a[mid]) {
           return bSearch(key, a, mid + 1, ub);
          }
     }
     return -1;
 }

void binarySearch() {
 int elem,flag;
 inputArray();
 printf("ENTER ELEMENT TO BE SEARCHED : ");
 scanf("%d",&elem);
 flag = bSearch(elem,a,0,n-1);
 if(flag < 0) {
     printf("ELEMENT NOT FOUND!");
     } else {
           printf("%d FOUND AT %d POSITION ON ARRAY",elem,flag);
     }
 }

void linearSearch() {
 int elem,i;
 inputArray();
 printf("ENTER ELEMENT TO BE SEARCHED : ");
 scanf("%d",&elem);
 for(i = 0 ; i < n ; i++) {
     if(elem == a[i]) {
      printf("%d FOUND AT %d POSITION OF ARRAY",elem,i);
      return;
      }
     }
     printf("ELEMENT NOT FOUND!");
 }

void swap(int * ip1, int * ip2) {
 int temp = * ip1;
 * ip1 = * ip2;
 * ip2 = temp;
}
```

```
void selectionSort() {
 int i,j;
 inputArray();
 for(i = 0 ; i < n - 1 ; i++) {
     for(j = i + 1 ; j < n ; j++) {
          if(a[i] > a[j]) {
           swap(&a[i],&a[j]);
          }
      }
     }
     displayArray();
}

void bubbleSort() {
 int i,j;
 inputArray();
 for(i = 0 ; i < n - 1 ; i++) {
     for(j = i ; j < n - i - 1; j++) {
      if(a[j] > a[j+1]) {
          swap(&a[j], &a[j+1]);
      }
     }
 }
 displayArray();
}

void insertionSort() {
 int i, key,j;
 inputArray();
 for(i = 1 ; i < n ; i++) {
     key = a[i];
     j   = i - 1;
     while(j >= 0 && a[j] > key) {
      a[j + 1] = a[j];
      j--;
     }
     a[j+1] = key;
 }
 displayArray();
}

int partition(int * a ,int lb, int ub) {
 int j = lb, i = j - 1, pivot = a[ub];
```

```
 while(j < ub) {
     if(a[j] <= pivot) {
      i++;
      swap(&a[i], &a[j]);
      }
      j++;
     }
     swap(&a[i+1], &a[ub]);
     return i+1;
 }


void quickSort(int * a, int lb, int ub) {
 int pivot;
 if(lb < ub) {
     pivot = partition(a,lb,ub);
     quickSort(a,lb,pivot-1);
     quickSort(a,pivot+1,ub);
 }
}

struct stacks {
     int stack[MAX];
     int top;
     }
     bucket[10], stack_main;

void initStack() {
 int i;
 for(i = 0 ; i < 10 ; i++) {
     bucket[i].top  = -1;
     }
     stack_main.top = -1;
 }

int popb(int place) {
 if(bucket[place].top == -1) {
     printf("BUCKET UNDERFLOW!");
     } else {
      return bucket[place].stack[bucket[place].top--];
      }
      return 0;
}
```

```
void pushb(int num,int place) {
 if(bucket[place].top == MAX - 1) {
     printf("BUCKET OVERFLOW");
     } else {
      bucket[place].stack[++bucket[place].top]  = num;
      }
}

void push(int num) {
if(stack_main.top == MAX - 1) {
 printf("STACK_MAIN OVERFLOW!");
 } else {
 stack_main.stack[++stack_main.top] = num;
 }
}

int pop() {
 if(stack_main.top == -1) {
     printf("STACK UNDERFLOW!");
     } else {
      return stack_main.stack[stack_main.top--];
      }
    return 0;
 }

void bucketSort() {
     int d,i,place = 1,num,temp = 0,passes = 0;
     initStack();
     inputArray();
     for(i = 0 ; i < n ; i++) {
      num = a[i];
      while(num > 0) {
          temp++;
          num /= 10;
      }
      if(temp > passes) {
           passes = temp;
      }
           temp = 0;
     }
     for(i = 0 ; i < n ; i++) {
      push(a[i]);
```

```
     }
     while(passes--) {
      while(stack_main.top != -1) {
          num = pop();
          d = (num % (10 * place))/place;
          pushb(num,d);
      }
     for(i = 9 ; i >= 0 ; i--) {
      while(bucket[i].top != -1) {
          push(popb(i));
           }
     }
      place *= 10;
 }
 i = 0;
 while(stack_main.top != -1) {
     a[i++] = pop();
 }
 displayArray();
}

void createHeap(int a[MAX],int n) {
 int i,j = 0,parent,child,temp;
 for(i = 0 ; i < n ; i++) {
     parent = (i - 1)/2;
     child  = i;
 while(a[parent] < a[child])  {
      swap(&a[parent],&a[child]);
      child  = parent;
      parent = (child - 1)/2;
    }
  }
}

void adjustHeap(int a[MAX], int n) {
int rchild,lchild,parent,i,j;
for(i = n - 1 ; i > 0 ; i--) {
 swap(&a[0], &a[i]);
 parent = 0;
 lchild = parent * 2 + 1;
 rchild = parent * 2 + 2;
 while(rchild < i || lchild < i) {
     if(a[lchild] < a[rchild] && rchild < i && a[parent] < a[rchild]) {
```

```
      swap(&a[parent], &a[rchild]);
      parent = rchild;
      } else if(a[parent] < a[lchild] && lchild < i) {
          swap(&a[parent], &a[lchild]);
          parent = lchild;
          } else break;
     lchild = parent * 2 + 1;
     rchild = parent * 2 + 2;
   }
 }
}


void main() {
 int ch;
 while(1) {
 clrscr();
 printf("SEARCHING AND SORTING IN C\n\n");
 printf("1. SEARCHING \n");
 printf("2. SORTING \n");
 printf("3. EXIT \n");
 printf("ENTER YOUR CHOICE :");
 scanf("%d",&ch);
 switch(ch) {
     case 1 : while(1) {
               clrscr();
               printf("SEARCHING\n\n");
               printf("1. LINEAR SEARCH\n");
               printf("2. BINARY SEARCH\n");
               printf("3. EXIT\n");
               printf("ENTER YOUR CHOICE : ");
               scanf("%d",&ch);
               switch(ch) {
                case 1 : linearSearch();
                         break;
                case 2 : binarySearch();
                         break;
                case 3 : exit(0);
               default : printf("WRONG CHOICE!");
                    }
               getch();
               }
     case 2 : while(1) {
```

```
                clrscr();
                printf("SORTING\n\n");
                printf("1. SELECTION SORT\n");
                printf("2. BUBBLE SORT\n");
                printf("3. INSERTION SORT\n");
                printf("4. QUICK SORT\n");
                printf("5. BUCKET SORT\n");
                printf("6. HEAP SORT\n");
                printf("7. EXIT\n");
                printf("ENTER YOUR CHOICE : ");
                scanf("%d",&ch);
                switch(ch) {
                 case 1 : selectionSort();
                          break;
                 case 2 : bubbleSort();
                          break;
                 case 3 : insertionSort();
                          break;
                 case 4 : inputArray();
                          quickSort(a,0,n-1);
                          displayArray();
                          break;
                 case 5 : bucketSort();
                          break;
                 case 6 : inputArray();
                          createHeap(a,n);
                          adjustHeap(a,n);
                          displayArray();
                          break;
                 case 7 : exit(0);
                default : printf("WRONG CHOICE!");
                 }
                getch();
                }
     case 3 : exit(0);
     default : printf("WRONG CHOICE!");
      }
   getch();
  }
}
```

OUTPUT

MAIN MENU

```
SEARCHING AND SORTING IN C

1. SEARCHING
2. SORTING
3. EXIT
ENTER YOUR CHOICE :
```

LINEAR SEARCH

```
SEARCHING

1. LINEAR SEARCH
2. BINARY SEARCH
3. EXIT
ENTER YOUR CHOICE : 1
ENTER SIZE OF ARRAY :5
ENTER ELEMENT 0 : 7
ENTER ELEMENT 1 : 9
ENTER ELEMENT 2 : 1
ENTER ELEMENT 3 : 15
ENTER ELEMENT 4 : 99
ENTER ELEMENT TO BE SEARCHED : 99
99 FOUND AT 4 POSITION OF ARRAY
```

BINARY SEARCH

```
SEARCHING

1. LINEAR SEARCH
2. BINARY SEARCH
3. EXIT
ENTER YOUR CHOICE : 2
ENTER SIZE OF ARRAY :10
ENTER ELEMENT 0 : 1
ENTER ELEMENT 1 : 5
ENTER ELEMENT 2 : 9
ENTER ELEMENT 3 : 10
ENTER ELEMENT 4 : 15
ENTER ELEMENT 5 : 29
ENTER ELEMENT 6 : 37
ENTER ELEMENT 7 : 46
ENTER ELEMENT 8 : 50
ENTER ELEMENT 9 : 61
ENTER ELEMENT TO BE SEARCHED : 46
46 FOUND AT 7 POSIITION ON ARRAY
```

SELECTION SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 1
ENTER SIZE OF ARRAY :6
ENTER ELEMENT 0 : 12
ENTER ELEMENT 1 : -1
ENTER ELEMENT 2 : 457
ENTER ELEMENT 3 : 1111
ENTER ELEMENT 4 : 3246
ENTER ELEMENT 5 : 2

ARRAY : -1 2 12 457 1111 3246 _
```

BUBBLE SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 2
ENTER SIZE OF ARRAY :6
ENTER ELEMENT 0 : -12
ENTER ELEMENT 1 : 457
ENTER ELEMENT 2 : 22
ENTER ELEMENT 3 : -90
ENTER ELEMENT 4 : 2345
ENTER ELEMENT 5 : 12

ARRAY : -12 -90 12 22 457 2345 _
```

INSERTION SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 3
ENTER SIZE OF ARRAY :6
ENTER ELEMENT 0 : 234
ENTER ELEMENT 1 : -12
ENTER ELEMENT 2 : 121
ENTER ELEMENT 3 : 4590
ENTER ELEMENT 4 : 212
ENTER ELEMENT 5 : -934

ARRAY : -934 -12 121 212 234 4590
```

QUICK SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 4
ENTER SIZE OF ARRAY :7
ENTER ELEMENT 0 : 1
ENTER ELEMENT 1 : 45
ENTER ELEMENT 2 : 0
ENTER ELEMENT 3 : -10
ENTER ELEMENT 4 : 243
ENTER ELEMENT 5 : 89
ENTER ELEMENT 6 : 667

ARRAY : -10 0 1 45 89 243 667
```

BUCKET SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 5
ENTER SIZE OF ARRAY :6
ENTER ELEMENT 0 : 123
ENTER ELEMENT 1 : 435
ENTER ELEMENT 2 : 1
ENTER ELEMENT 3 : 9
ENTER ELEMENT 4 : 2456
ENTER ELEMENT 5 : 67

ARRAY : 1 9 67 123 435 2456
```

HEAP SORT

```
SORTING

1. SELECTION SORT
2. BUBBLE SORT
3. INSERTION SORT
4. QUICK SORT
5. BUCKET SORT
6. HEAP SORT
7. EXIT
ENTER YOUR CHOICE : 6
ENTER SIZE OF ARRAY :6
ENTER ELEMENT 0 : 123
ENTER ELEMENT 1 : -567
ENTER ELEMENT 2 : 0
ENTER ELEMENT 3 : 1
ENTER ELEMENT 4 : -2
ENTER ELEMENT 5 : 1111

ARRAY : -567 -2 0 1 123 1111 _
```

END OF SEARCHING & SORTING