Exercise 5: Arrays

Nanda H Krishna

1 March 2018

1 Binary Search

Problem description: We are given a sorted array of numbers. Define a function `binary_search(a, n, target)` that searches for `target` in `a[0:n]` using binary search algorithm. Let the function return an index `i` such that `a[0:i] < target <= a[i:n]`.

Specification: The function `binary_search()` takes a sorted array `a`, the length `n` and the element to be searched `t` as inputs and returns the index `i` to the calling function.

Prototype:

```
int binary_search(int a[], int n, int t)
```

Program design: The program consists of a function `binary_search(int a[], int n, int t)` which returns the index of the element to find, and `main()`, which gets input from `stdin`, calls the function and prints the value on `stdout`.

Algorithm: The algorithm for binary search is as follows:

```
binary_search(a[], t, n):
  l, u = 0, n-1
  while l <= u:
    m = (l + u)//2
    if a[m] < t:
      l = m + 1
    else if a[m] > t:
      u = m
    else:
      return m
  return -1
```

Program:

```
#include<stdio.h>
int binarysearch(int a[], int n, int t)
{
  int l = 0, u = n - 1;
  int mid;
  while(l <= u) {
    mid = (l + u)/2;
    if(a[mid] == t) return mid; // Found
    else if(a[mid] > t) u = mid - 1;
    else l = mid + 1;
  }
  return -1; // Not found
}
int main()
{
  int a[10], t;
  for(int i = 0; i < 10; i++) scanf("%d", &a[i]);
  scanf("%d", &t);
  int r = binarysearch(a, 10, t);
  if(r == -1) printf("Not found");
  else printf("Found at index %d", r);
  return 0;
}
```

Test Input:

```
0 1 2 3 4 5 6 7 8 9 5
```

Output:

```
Found at index 5
```

2 Selection Sort

Problem description: Implement selection sort and test the function from `main()` for several lists of numbers.

Specification: The function `minimum()` takes `a`, `low` and `high` as parameters and returns index of smallest element. The function `selection_sort()` sorts the array `a` in ascending order.

Prototype:

```
int minimum(int a[], int low, int high)
void selection_sort(int a[], int n)
```

Program design: Function `minimum()` takes array `a`, start index `low`, and end index `high` as inputs and returns the index of smallest number, and `selection_sort()` takes array `a`, length `n` as inputs and sorts the array in ascending order. Testing is done from `main()`.

Algorithm: The algorithm for selection sort is as follows:

```
selection_sort(a[], n):
  for i in range(n):
    m = minimum(a, i, n)
    a[i], a[m] = a[m], a[i]
```

Program:

```
#include<stdio.h>
int minimum(int a[], int low, int high)
{
  int i = low, min = low;
  while(i < high) {
    if(a[i] < a[min]) min = i;
    i++;
  }
  return min;
}
void selection_sort(int a[], int n)
{
  for(int i = 0; i < n - 1; i++) {
    int s = minimum(a, i, n);
    int t = a[s];
    a[s] = a[i];
    a[i] = t;
  }
}
int main()
{
  int a[10];
  for(int i = 0; i < 10; i++) scanf("%d", &a[i]);
  selection_sort(a, 10);
  for(int i = 0; i < 10; i++) printf("%d ", a[i]);
  return 0;
}
```

Test Input:

```
5 6 7 9 2 3 4 0 1 8
1 1 1 1 1 1 1 1 1 1
-1 -2 -3 -4 0 1 2 3 4 0
```

Output:

```
0 1 2 3 4 5 6 7 8 9
1 1 1 1 1 1 1 1 1 1
-4 -3 -2 -1 0 0 1 2 3 4
```

3 Polish National Flag

Problem description: In an array of items a[low:high], each item is either positive or negative. Define a function pnf(a, low, high) that partitions the array into two subarrays a[low:i] and a[i:high] such that all the negative items of the array form [low:i], and all the positive items form [i:high].

Specification: Function pnf() takes array a, low, high as input and returns the index of the last negative number in the new array.

Prototype:

```
int pnf(int a[], int l, int h)
```

Program design: The program has a function pnf(int a[], int l, int h) which performs the partitioning of the array into negative and positive portions, swap(int s[], int a, int b) to swap elements, print_array(int a[], int n) to print the array and main(), which gets the input from stdin, calls the pnf() and prints the result on stdout.

Algorithm: The algorithm for PNF is as follows:

```
pnf(a[], low, high):
  mid = 0
  while mid <= high:
    if a[mid] < 0:
      swap(a, low++, mid++)
    else if a[mid] >= 0:
      mid++
  return mid
```

Program:

```
#include<stdio.h>
void swap(int s[], int a, int b)
{
    int temp = s[a];
    s[a] = s[b];
    s[b] = temp;
}
int pnf(int a[], int low, int high)
{
    int mid = 0;
    while (mid <= high) {
      if(a[mid] == -1) swap(a, low++, mid++);
      else if(a[mid] == 1) mid++;
    }
    return mid;
}
void print_array(int a[], int n)
```

```
{
    int i;
    for(i = 0; i < n; i++) printf("%d ", a[i]);
}
int main()
{
    int a[10];
    int n = 10;
    for(int i = 0; i < 10; i++) scanf("%d", &a[i]);
    pnf(a, 0, n - 1);
    print_array(a, n);
    return 0;
}
```

Test Input:

```
1 1 -1 1 -1 1 -1 1 1 -1
-20 10 30 40 -50 -60 70 80 -90 20
2 2 2 2 2 2 2 2 2 2
```

Output:

```
-1 -1 -1 -1 1 1 1 1 1 1
-20 -50 -60 -90 10 30 40 70 80 20
2 2 2 2 2 2 2 2 2 2
```

4 Dutch National Flag

Problem description: Similar to PNF, partition the array `a` into three subarrays `[l:i]`, `[i:j]` and `[j:h]`. Each item of the array has one of three properties. Items having the same property should form one subarray each.

Specification: Function `dnf()` takes array `a`, `low`, `high` as input and results in the array being split into three subarrays, each with a certain shared property.

Prototype:

```
void dnf(int a[], int low, int high)
```

Program design: The program has a function `dnf(int a[], int l, int h)` which performs the partitioning of the array into negative, zero and positive portions, `swap(int s[], int a, int b)` to swap elements, `print_array(int a[], int n)` to print the array and `main()`, which gets the input from `stdin`, calls the `dnf()` and prints the result on `stdout`.

Algorithm: The algorithm for DNF is as follows:

```
dnf(a, l, h):
  mid = 0
  while mid <= high:
    if a[mid] < 0:
      swap(a, low++, mid++)
    else if a[mid] is 0:
      mid++
    else:
      swap(a, mid, high--)
```

Program:

```
#include<stdio.h>
void swap(int s[], int a, int b)
{
    int temp = s[a];
    s[a] = s[b];
    s[b] = temp;
}
void dnf(int a[], int low, int high)
{
    int mid = 0;
    while (mid <= high) {
if(a[mid] < 0)
  swap(a, low++, mid++);
else if(a[mid] == 0)
  mid++;
else
  swap(a, mid, high--);
    }
}
void print_array(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
printf("%d ", a[i]);
}
int main()
{
    int a[10];
    int n = 10;
    for(int i = 0; i < n; i++) scanf("%d", &a[i]);
    dnf(a, 0, n - 1);
    print_array(a, n);
    return 0;
}
```

Test Input:

```
1 1 0 1 -1 1 -1 0 0 -1
-3 -3 -3 -3 -3 -3 -3 -3 -3 -3
```

```
0 10 20 -30 -40 50 -60 70 -80 90
```

Output:

```
-1 -1 -1 0 0 0 1 1 1 1
-3 -3 -3 -3 -3 -3 -3 -3 -3 -3
-30 -40 -60 -80 0 10 20 50 70 90
```