CPE102 Programming II

Week 7

String Functions
Sorting & Searching

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

String

- A string is an array of characters terminated by the NULL char '\0'
- Example:
  - char str[8];
  - Declares a char array that contains at most 8 chars.
- If char array str will be used to store strings, it can contain at most 7 chars and MUST be terminated by the NULL char '\0'

String functions

- C standard library provides many functions to manipulate strings.
- You need to include <string.h> to use these functions
  #include<string.h>
- Here are some of the important functions:
  - strcpy(char *dest, const char *src);
    - Copies the string pointed to, by src to dest.
  - strcat(char *dest, const char *src);
    - Appends the string pointed to, by src to the end of the string pointed to by dest.
  - strlen(const char *str);
    - Computes the length of the string str up to but not including the terminating null character.
  - strcmp(const char *str1, const char *str2);
    - Compares the string pointed to, by str1 to the string pointed to by str2.

strcpy() function

StringCopy.c

```c
#include <stdio.h>
#include <string.h>

int main () {
    char src[40];
    char dest[100];

    strcpy(src, "This is the source string");
    strcpy(dest, src);

    printf(" Final copied string(destination) : %s\n", dest);
    getch();
    return(0);
}
```

```
Final copied string(destination) : This is the source string
```

strcpy() function code

```
char *strcpy(char *str1, const char *str2)
{
  char *p = str1;

  while (*str2)
      *p++ = *str2++;

  *p = '\0';
  return str1;
} /* end-strcpy */
```

strlen() function

StringLength.c

```c
#include <stdio.h>
#include <string.h>

int main () {
    char str[50]="What is my length?";
    int len;

    len = strlen(str);
    printf(" Length of |%s| is |%d|\n", str, len);

    return(0);
}
```

```
Length of |What is my length?| is |18|
```

strlen() function code

```
int strlen(const char *str)
{
  int len = 0;

  while(*str++)
      len++;

  return len;
} /* end-strlen */
```

strcat() function

StringConcatenate.c

```c
#include <stdio.h>
#include <string.h>

int main () {
   char src[50], dest[50];

  strcpy(dest, "This is destination,");
  strcpy(src,  "This is source");

   strcat(dest, src);

   printf("  Final destination string : |%s|", dest);
   getch();
   return(0);
}
```

```
Final destination string : |This is destination,This is source
```

strcat() function code

```
char *strcat(char *str1, const char *str2)
{
  char *p = str1;

  while(*p)
      p++;

  while(*str2)
      *p++ = *str2++;

  *p = '\0';
  return str1;
} /* end-strcat */
```

strcmp() function

StringCompare.c

```c
#include <stdio.h>
#include <string.h>

int main () {
   char str1[15], str2[15];
   int ret;
   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");

   ret = strcmp(str1, str2);

   if(ret < 0) {
      printf(" str1 is less than str2");
   } else if(ret > 0) {
      printf(" str2 is less than str1");
   } else {
      printf(" str1 is equal to str2");
   }
   getch();
   return(0);
}
```

```
str2 is less than str1
```

strcmp() function code

```
int strcmp(const char *str1, const char *str2)
{
  while (*str1 && *str2 && *str1 == *str2){
    str1++; str2++;
  } /* end-while */

  return *str1-*str2;
} /* end-strcmp */
```

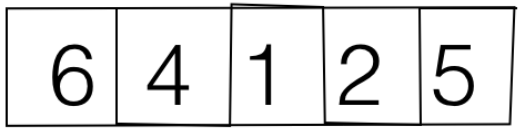
Sorting

- Placing a group of data in descending or ascending order.
- Sorting data is very useful for computer systems.
- Makes searching and listing a group of data faster and easier.
- Most popular sorting algorithms:
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Quick sort
- Time complexity: is the number of operations an algorithm performs to complete its task. The algorithm that performs the task in the smallest number of operations is considered the most efficient one.

Bubble Sort

- Time complexity is $O(n^2)$
- Design of algorithm is easy, but algorithm is not efficient.
- Can be used for small size lists or lists having mostly sorted items

Bubble Sort



Iteration 1:	Iteration 2:	Iteration 3:	Iteration 4:
6 4 1 2 5	**4 1** 2 5 6	**1 2** 4 5 6	**1 2** 4 5 6
4 **6 1** 2 5	1 **4 2** 5 6	1 **2 4** 5 6	1 2 4 5 6
4 1 **6 2** 5	1 2 **4 5** 6	1 2 4 5 6	
4 1 2 **6 5**	1 2 4 5 6		
4 1 2 5 6			

Bubble Sort

bubble_sort.c

```c
#include <stdio.h>
void bubble_sort(long [], long);

int main()
{
  long array[100], n, c;

  printf("Enter number of elements\n");
  scanf("%ld", &n);

  printf("Enter %ld integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%ld", &array[c]);

  bubble_sort(array, n);

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
     printf("%ld\n", array[c]);

  return 0;
}
```

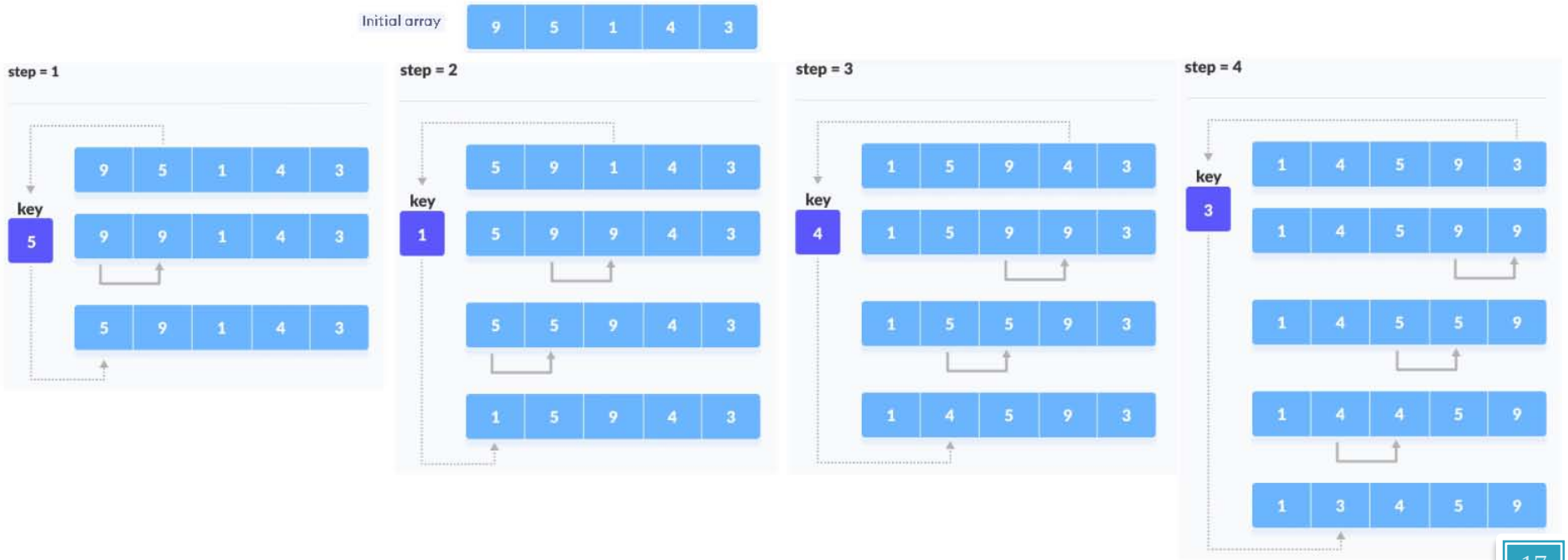
```c

void bubble_sort(long list[], long n)
{
  long c, d, t;

  for (c = 0 ; c < n - 1; c++) {
    for (d = 0 ; d < n - c - 1; d++) {
      if (list[d] > list[d+1]) {
        /* Swapping */
        t          = list[d];
        list[d]    = list[d+1];
        list[d+1] = t;
      }
    }
  }
}
```

Insertion Sort

- Appropriate for inserting an item into an already sorted list of data.
- Complexity of inserting an item into an already sorted list of data: O (n)
- If list or array is not sorted complexity: $O(n^2)$

Insertion Sort

At each iteration it insert the element at its correct location and this process is repeated until complete array is sorted.



Insertion Sort

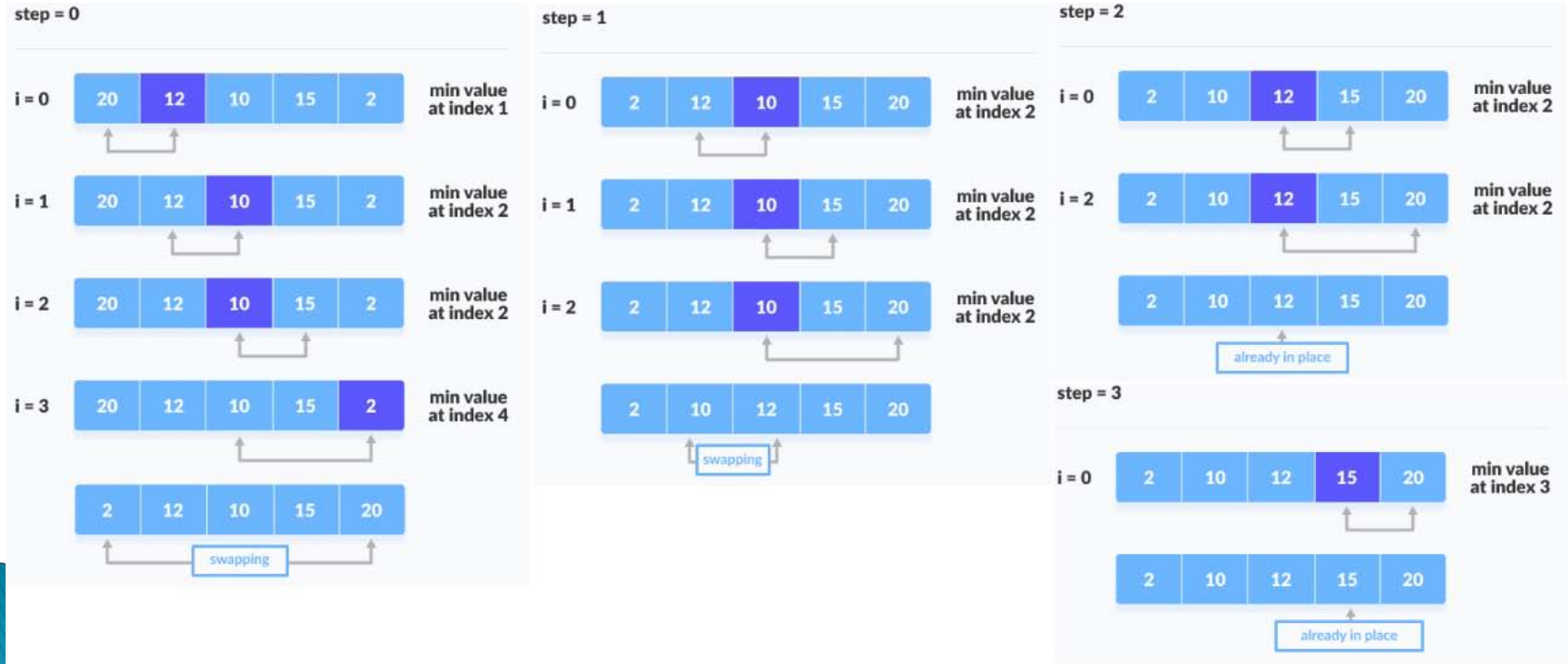
insertionSort.c

```c

#include <stdio.h>



// Driver code
int main() {
  int data[] = {9, 5, 1, 4, 3};
  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);

  printf("Sorted array in ascending order:\n");
  printArray(data, size);
}



// Function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    printf("%d ", array[i]);
  }
  printf("\n");
}
```

```c



void insertionSort(int array[], int size) {

  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;

    // Compare key with each element on the left of it
    // until an element smaller than it is found.
    // For descending order, change key<array[j] to key>array[j].
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}
```

Selection Sort

- If an item is in its true place it does not change its order.
- Take the first item in the list and exchange with the minimum item of others. Repeat this until the last item.



Selection Sort

selectionSort.c

```c

#include <stdio.h>

// driver code
int main() {
  int data[] = {20, 12, 10, 15, 2};
  int size = sizeof(data) / sizeof(data[0]);
  selectionSort(data, size);
  printf("Sorted array in Acsending Order:\n");
  printArray(data, size);
}



// function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

// function to swap the the position of two elements
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;

    for (int i = step + 1; i < size; i++) {

      // To sort in descending order, change > to < in this line.
      // Select the minimum element in each loop.
      if (array[i] < array[min_idx])
        min_idx = i;
    }

    // put min at the correct position
    swap(&array[min_idx], &array[step]);
  }
}
```

Search

- The process of finding a particular element of an array is called searching.
- Two searching techniques will be discussed
  - Linear Search
  - Binary Search

Linear Search

- Compares each element of the array with the search key.
- Since the array is not in any particular order, it is just as likely that the value will be found in the first element as in the last.
- In the worst case with N number of elements, the algorithm's complexity is O(N)
- It should not be used in large size arrays.

Linear Search

LinearSearch.c

```c
#include <stdio.h>

int search(int array[], int n, int x) {

  // Going through array sequentially
  for (int i = 0; i < n; i++)
    if (array[i] == x)
      return i;
  return -1;
}

int main() {
  int array[] = {2, 4, 0, 1, 9};
  int x = 1;
  int n = sizeof(array) / sizeof(array[0]);

  int result = search(array, n, x);

  (result == -1) ? printf("Element not found") : printf("Element found at index: %d", result);
}
```

Thanks ☺