Data structures and algorithms

Tutorial 5

Amr Keleg

Faculty of Engineering, Ain Shams University

March 25, 2019

Contact: amr_mohamed@live.com

Outline

Outline

Why this code compile and run?

```
#include<iostream>
using namespace std;

int main(){
   int size;
   cin>>size;
   int arr[size];
}
```

Why this code compile and run?

```
#include<iostream>
using namespace std;

int main(){
   int size;
   cin>>size;
   int arr[size];
}
```

- Yes, If you are using codeblocks (g++/gcc compilers)
- No, If you are using Visual Studio

Outline

VLA (Variable Length Arrays)

- Allows the creation of arrays in the stack whose size is determined at run-time.
- treated like local variables. The compiler deletes them when the function terminates.
- Not part of the standard C++ specification.

VLA (Variable Length Arrays)

- g++/gcc have extensions that allow the usage of VLA.
- g++ vla.cpp -o vla.exe
- Disable the extension and try again: g++ -Wvla vla.cpp -o vla.exe
- The Linux Kernel Is Now VLA-Free: A Win For Security, Less Overhead & Better For Clang.
- `https://www.phoronix.com/scan.php?page=news_item&px=Linux-Kills-The-VLA`

Outline

Write a function that takes and array and mutliplies each element by 2.

```
int * double_arr(int arr[], int size){
  int result[size];
  for(int i=0; i<size; i++){
    result[i] = 2 * arr[i];
  }
  return result;
}
```

Problem: result is local to the function and will be deleted once the function ends.

```
int * double_arr(int arr[], int size){
  for(int i=0; i<size; i++){
    arr[i] = 2* arr[i];
  }
  return arr;
}
```

The code works since the array won't be deleted when the function execution ends.

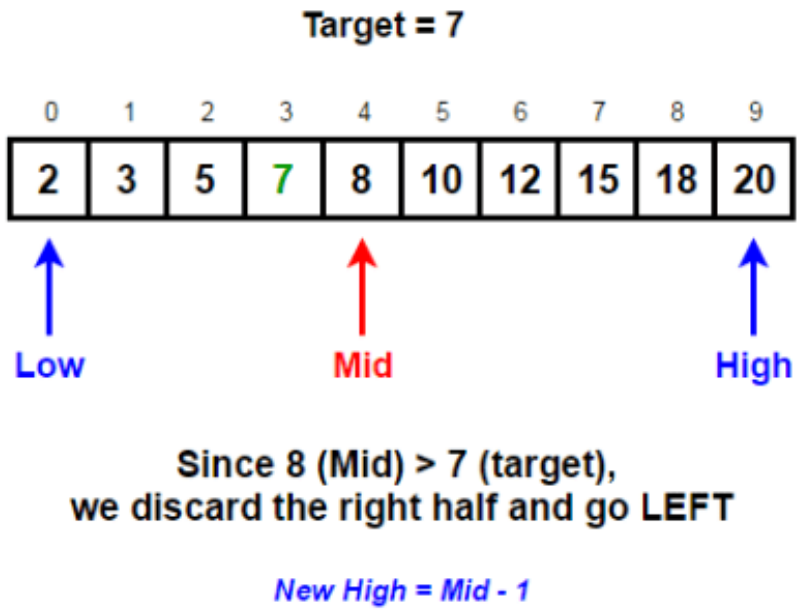
```
void double_arr(int arr[], int size){
  for(int i=0; i<size; i++){
    arr[i] = 2* arr[i];
  }
}
```

We can just make a void function.

Outline

Outline

In computer science, binary search, also known as half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array.



```
int binary_search(int arr[], int item, int l, int r){
  if (l==r){
    if (arr[l] != item) return -1;
    return l;
  }

  int mid = (l+r)>>1;
  if (arr[mid] == item)
    return mid;
  if(arr[mid] >item)
    return binary_search(arr, item, l, mid-1);
  // arr[mid] < item
  return binary_search(arr, item, mid+1, r);
}
```

Outline

We will discuss it later.
May be the tutorial after the midterms :D

Outline

Outline

Write a function that sorts an array given that its two halves are sorted.
What is the expected complexity?

```
void merge(int arr[], int size)
{
   ...
}

int main(){
   int arr[6] = {3, 5, 7, 1, 2, 6};
   merge(arr, 6);
}
```

Write a function that sorts an array given that its two halves are sorted.

```
void merge(int arr[], int size)
{
  int * result = new int[size];
  int index = 0;
  int l_index = 0, l_limit = size/2;
  int r_index = size/2, r_limit = size;
  while(l_index < l_limit && r_index < r_limit){
    if (arr[l_index] <= arr[r_index])
      result[index++] = arr[l_index++];
    else
      result[index++] = arr[r_index++];
  }

  // Copy the remaining elements of either the left or t
}
```

Write a function that sorts an array given that its two halves are sorted.

```
void merge(int arr[], int size)
{
  // Copy the remaining elements of either the left or t
  while(l_index < l_limit)
    result[index++] = arr[l_index++];
  while(r_index < r_limit)
    result[index++] = arr[r_index++];

  for(int i=0; i< size; i++)
    arr[i] = result[i];

  delete [] result;
}
```

Write a function that sorts an array given that its two halves are sorted.

```
void merge(int arr[], int size, int result[])
{
   ...
}
int main(){
   int arr[6] = {3, 5, 7, 1, 4, 6};
   int res[6];
   merge(arr, 6, res);
}
```

Write a function that sorts an array given that its two halves are sorted.

```
void merge(int arr[], int size, int result[])
{
  int index = 0;
  int l_index = 0, l_limit = size/2;
  int r_index = size/2, r_limit = size;
  while(l_index < l_limit && r_index < r_limit){
    if (arr[l_index] <= arr[r_index])
      result[index++] = arr[l_index++];
    else
      result[index++] = arr[r_index++];
  }

  // Copy the remaining elements of either the left or t
}
```

Write a function that sorts an array given that its two halves are sorted.

```
void merge(int arr[], int size, int result[])
{
  // Copy the remaining elements of either the left or t
  while(l_index < l_limit)
    result[index++] = arr[l_index++];
  while(r_index < r_limit)
    result[index++] = arr[r_index++];
}
```

```
void merge(int l_arr[], int r_arr[],
        int l_size, int r_size, int sorted_arr[]){ ... }

void merge_sort(int arr[],
         int l, int r, int sorted_arr[]){
  if(l==r){
    sorted_arr[0] = arr[l];
    return ;
  }
  int mid = (l+r)/2;
  int *l_arr = new int[mid-l+1];
  int *r_arr = new int[r-mid];
  merge_sort(arr, l, mid, l_arr);
  merge_sort(arr, mid+1, r, r_arr);
  merge(l_arr, r_arr, mid-l+1, r-mid, sorted_arr);
  delete [] l_arr;
  delete [] r_arr;
}
```

Outline

```
void quick_sort(int arr[], int l, int r){
  if (l>=r) return;
  int pivot_val = arr[l];
  int i = l+1;
  int j = r;
  while(i<=j){
    while(arr[i] <= pivot_val) i++;

    while(arr[j] > pivot_val) j--;

    if (i<j)
      swap(arr[i], arr[j]);
  }
  swap(arr[l], arr[j]);
  quick_sort(arr, l, j-1);
  quick_sort(arr, i, r);
}
```

Outline

Outline

```
class Queue{
  public:
    Queue();
    int front();
    int back();
    void push(int v);
    void pop();
};
```

How to implement a queue?

- An array of fixed size
- A circular array
- A dynamically allocated array
- A linked list

Check the Dr.'s slides.

Outline

- Using a queue of jobs where each job has an estimated service time, Develop a method to compute the required service time of all jobs in the queue and the average waiting time per job.
- Try not to destroy the queue after finding the sum of the jobs.
- Apply your method on the following queue of four jobs.

```
void job_statisitics(queue q){
  int q_size = q.size();
  int total_service_time = 0;
  int total_waiting_time = 0;
  for (int i=0; i<q_size; i++){
    int cur_time = q.front();
    q.pop();
    total_waiting_time += total_service_time;
    total_service_time += cur_time;
    q.push(cur_time);
  }
  cout<< "Total time "<<total_service_time <<"\n";
  cout<< "Average waiting time "
    <<1.0 * total_waiting_time / q_size<<"\n";
}
```

Feedback form: `https://goo.gl/forms/vmVWf0JtNA1TLXRf2`