Time Complexity

Zhanfu Yang

How?

Time Measure Method

[1] Wall time: is simply the **total time elapsed** during the measurement. It's the time you can measure with a stopwatch, assuming that you are able to start and stop it exactly at the execution points you want.

[2] CPU time: **CPU time is how many seconds the CPU was busy**.

The Wall time:

The CPU time:

$$\text{CPU Time} = \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

1 Time Measure Method Wall time

[1] Chrono in C++. https://cplusplus.com/reference/chrono/

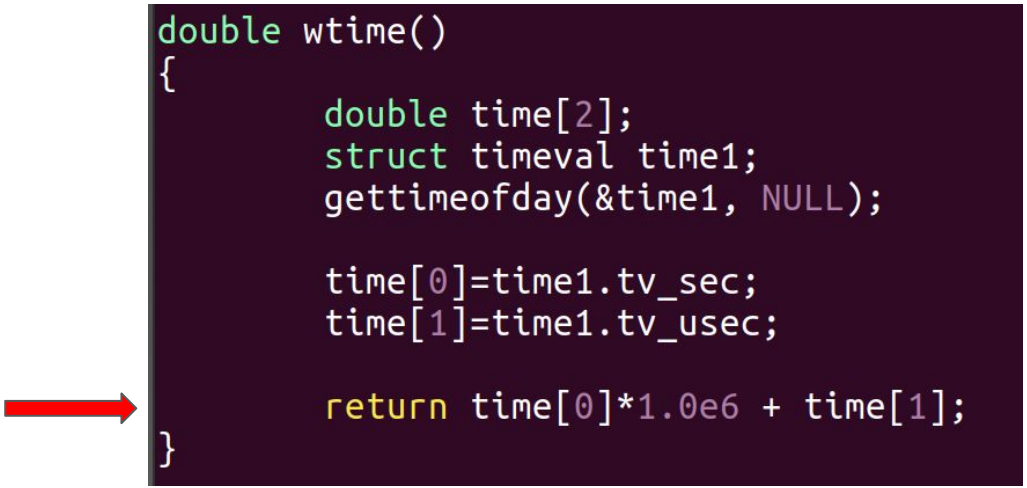
```
auto start = std::chrono::high_resolution_clock::now();
// void qsort(void *base, size_t nmemb, size_t size,
//                  int (*compar)(const void *, const void *));
//sort the array
qsort(array, size, sizeof(int), my_compare_func);

auto end = std::chrono::high_resolution_clock::now();
```

[2] Print out:

```
auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

auto duration2 = std::chrono::duration_cast<std::chrono::microseconds>(end2 - end);

auto duration3 = std::chrono::duration_cast<std::chrono::microseconds>(start - start0);

std::cout << "Time taken by normal search: " << duration3.count() << " microseconds" << std::endl;
std::cout << "Time taken by function quick sort: "  << duration.count() << " microseconds" << std::endl;
std::cout << "Time taken by function binary search: " << duration2.count() << " microseconds" << std::endl;
```

2 Time Measure Method Wall time

[1] Define time.
[2] timeval.
[3] Return will be the (us).



```
double wtime()
{
        double time[2];
        struct timeval time1;
        gettimeofday(&time1, NULL);

        time[0]=time1.tv_sec;
        time[1]=time1.tv_usec;

        return time[0]*1.0e6 + time[1];
}
```

3 Time Measure Method CPU time

[1] Clock();
[2] Record the start time.

```
double time_begin = wtime();
std::clock_t c_start = std::clock();

qsort(array, size, sizeof(int), my_compare_func);

// Insert the applications we want to measure the time consumption
// ...
// Matrix multiplication
// Qsort
// Binary Search
// Linear Search

double my_time = wtime() - time_begin;
std::clock_t c_end = std::clock();

std::cout << "The time consumption is: " << my_time << " (us).\n";
long double time_elapsed_ms = 1000000.0 * (c_end-c_start) / CLOCKS_PER_SEC;
std::cout << "The CPU time used: " << time_elapsed_ms << " (us).\n";
```

4 Time Measure Method CPU time

[1] Clock();
[2] Record the start time.
[3] Record the end time.

```
double time_begin = wtime();
std::clock_t c_start = std::clock();

qsort(array, size, sizeof(int), my_compare_func);

// Insert the applications we want to measure the time consumption
// ...
// Matrix multiplication
// Qsort
// Binary Search
// Linear Search

double my_time = wtime() - time_begin;
std::clock_t c_end = std::clock();

std::cout << "The time consumption is: " << my_time << " (us).\n";
long double time_elapsed_ms = 1000000.0 * (c_end-c_start) / CLOCKS_PER_SEC;
std::cout << "The CPU time used: " << time_elapsed_ms << " (us).\n";
```

5 Time Measure Method CPU time

[1] Clock();
[2] Record the start time.
[3] Record the end time.
[4] Transfer to the (us).

```
double time_begin = wtime();
std::clock_t c_start = std::clock();

qsort(array, size, sizeof(int), my_compare_func);

// Insert the applications we want to measure the time consumption
// ...
// Matrix multiplication
// Qsort
// Binary Search
// Linear Search

double my_time = wtime() - time_begin;
std::clock_t c_end = std::clock();

std::cout << "The time consumption is: " << my_time << " (us).\n";
long double time_elapsed_ms = 1000000.0 * (c_end-c_start) / CLOCKS_PER_SEC;
std::cout << "The CPU time used: " << time_elapsed_ms << " (us).\n";
```

Normal Search O(n)

```
int normal_search(int *array, int search_key, int beg, int end) {
        for (int i = beg; i < end; i ++) {
                if (array[i] == search_key) {
                        return i;
                }
        }
        return -1;
}
```

Binary Search O(log(n))

```
int binary_search(int *array, int search_key, int beg, int end)
{

        while(beg<=end)
        {
                int mid = (beg + end)/2;
                if(array[mid] == search_key)
                {
//                      printf("Found %d at index = %d\n",search_key,mid);
                        return 1;
                }else if(array[mid] < search_key){
                        beg = mid + 1;

                }else{
                        end = mid - 1;
                }
                //searching range changed to
//              printf("Valid range: array[%d] = %d -- array[%d] = %d for search_key = %d.\n", beg,array[beg],end,array[end-1], search_key);
        }

//      printf("Didn't find %d in this array\n", search_key);
        return -1;
}
```

Quick sort O(n*log(n))

```
double time_begin = wtime();
std::clock_t c_start = std::clock();

qsort(array, size, sizeof(int), my_compare_func);
```

Bubble sort O(n*n=n^2)

```
void bubble_sort(int *array, int beg, int end) {
        for (int i = beg; i < end; i ++) {
                for (int j = i+1; j < end; j ++) {
                        if(array[i] > array[j]) {
                                int tmp = array[j];
                                array[j] = array[i];
                                array[i] = tmp;
                        }
                }
        }
}
```

Matrix Multiplication O(n^3)

```
void matmul(int **A, int **B, int **C, int n, int k, int m)
{
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < m; j ++)
        {
            C[i][j] = 0;
            int tmp = 0;
            // **** Fill the content here for multiplication (4)
            for (int p = 0 ; p < n; p ++) {
                tmp += A[i][p]*B[p][j];
            }

            // End of the multiplication
        }
}
```

Time complexity analysis