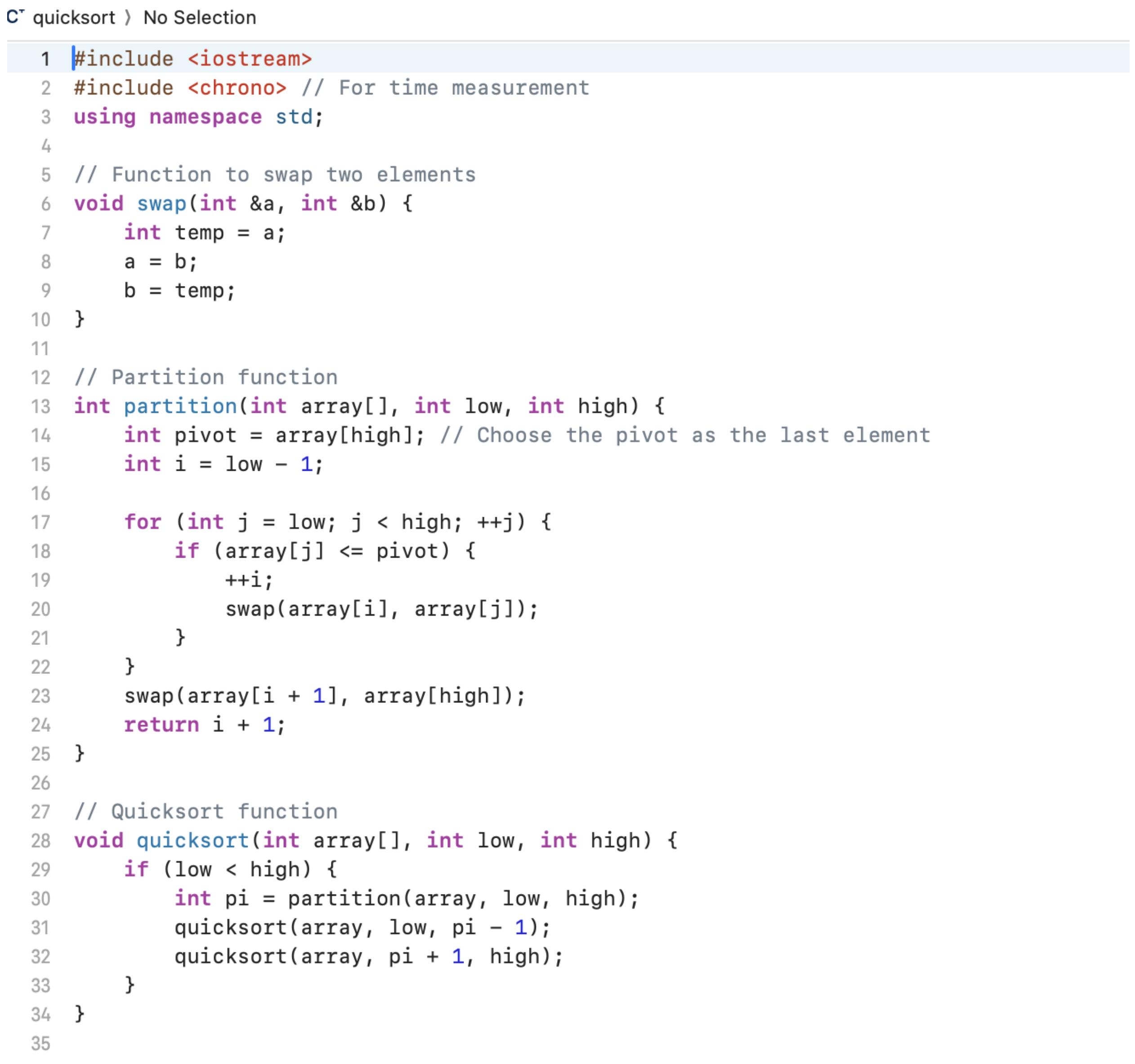
Bajwa, Qaasid 12/04/24

CSC21000 - QuickSort Algorithm

Section 1: Overview

The primary objective of this homework assignment was to create a quick sort code for an array of 10 integers or less that worked properly and measured the execution time. Then, we had to demonstrate our understanding of quick sort by doing it manually on paper for a small array.

Section 2: Code



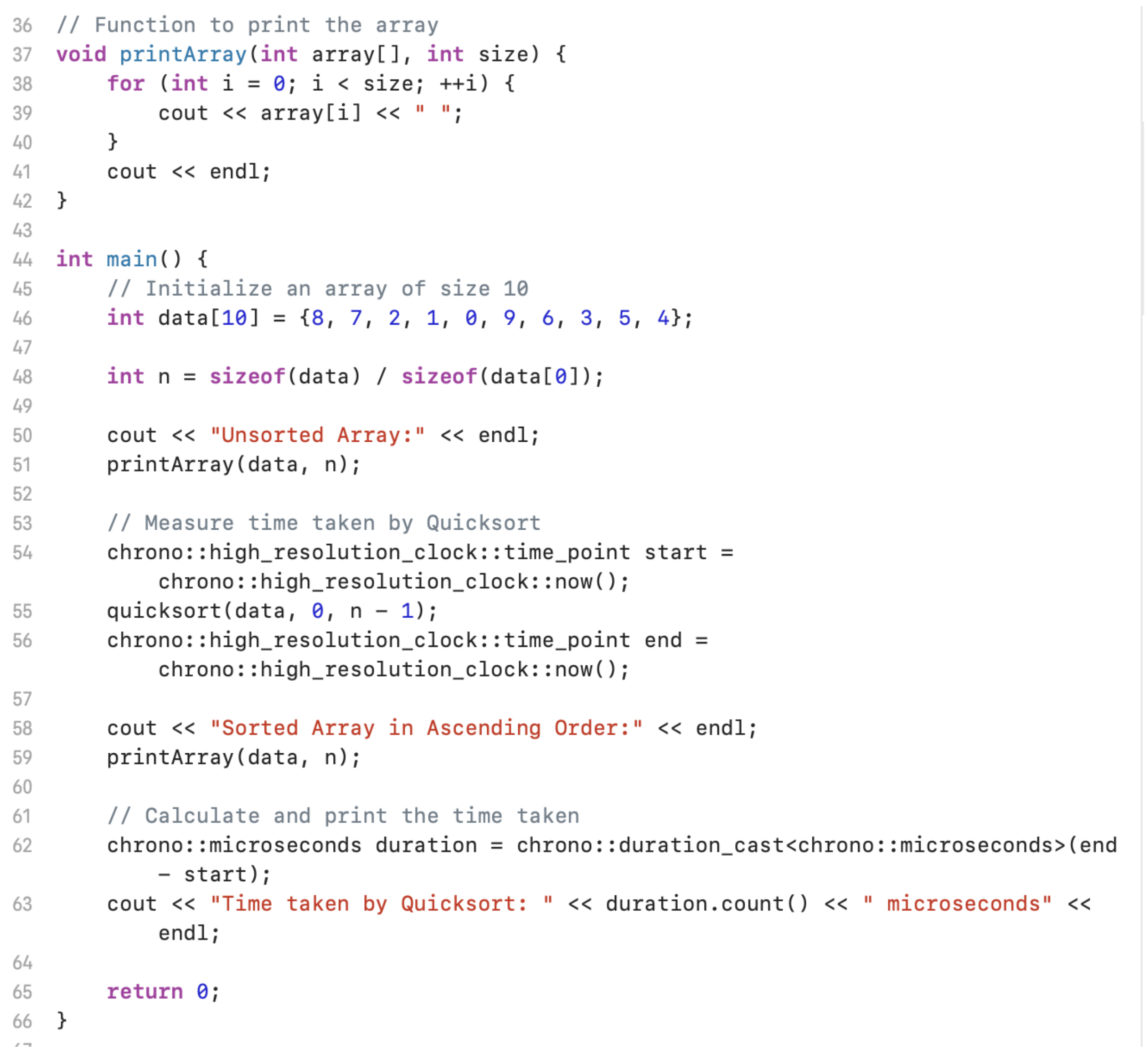
```cpp
#include <iostream>
#include <chrono> // For time measurement
using namespace std;

// Function to swap two elements
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

// Partition function
int partition(int array[], int low, int high) {
    int pivot = array[high]; // Choose the pivot as the last element
    int i = low - 1;

    for (int j = low; j < high; ++j) {
        if (array[j] <= pivot) {
            ++i;
            swap(array[i], array[j]);
        }
    }
    swap(array[i + 1], array[high]);
    return i + 1;
}

// Quicksort function
void quicksort(int array[], int low, int high) {
    if (low < high) {
        int pi = partition(array, low, high);
        quicksort(array, low, pi - 1);
        quicksort(array, pi + 1, high);
    }
}
```



```cpp
// Function to print the array
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
}

int main() {
    // Initialize an array of size 10
    int data[10] = {8, 7, 2, 1, 0, 9, 6, 3, 5, 4};

    int n = sizeof(data) / sizeof(data[0]);

    cout << "Unsorted Array:" << endl;
    printArray(data, n);

    // Measure time taken by Quicksort
    chrono::high_resolution_clock::time_point start =
        chrono::high_resolution_clock::now();
    quicksort(data, 0, n - 1);
    chrono::high_resolution_clock::time_point end =
        chrono::high_resolution_clock::now();

    cout << "Sorted Array in Ascending Order:" << endl;
    printArray(data, n);

    // Calculate and print the time taken
    chrono::microseconds duration = chrono::duration_cast<chrono::microseconds>(end
        - start);
    cout << "Time taken by Quicksort: " << duration.count() << " microseconds" <<
        endl;

    return 0;
}
```

This is my C++ code.

The swap function exchanges the values of two integers. It is used in the partition step to rearrange elements in the array.

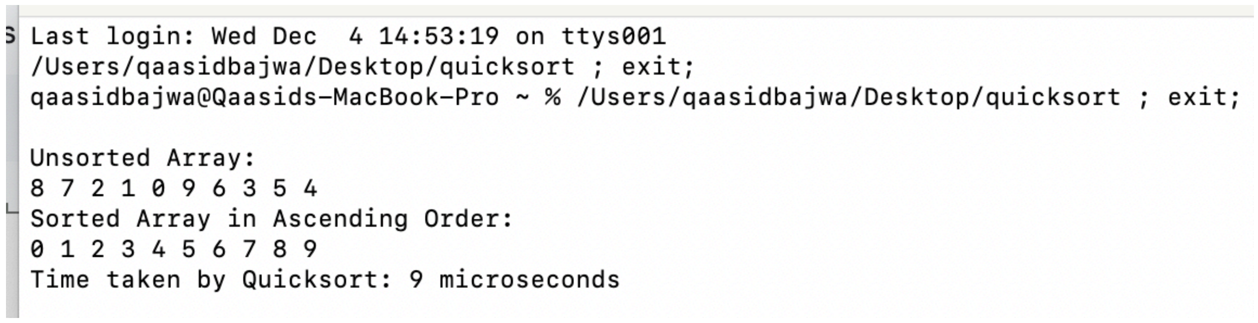
Next is the partition function. It divides the array into two parts based on the pivot. The pivot is selected as the last element in the array (array [high]). Two indices are used (i and j). i tracks the boundary of smaller and elements while j iterates through the array to compare elements with the pivot. If an element (array[j]) is smaller than or equal to the pivot, it is swapped with array[i + 1]. After the loop, the pivot is placed in its correct sorted position by swapping it with array[i + 1]. The purpose of the partition function is that it rearranges the array such that elements smaller than or equal to the pivot are on the left, while elements greater than the pivot are on the right.

The quicksort function recursively applies the Quicksort algorithm to the array. The partition function determines the pivot's correct position in the sorted array. The array is divided into two subarrays: the left which are elements before the pivot, and the right, which are elements after the pivot. Quicksort is called recursively on the left and right subarrays. The recursion stops when low is greater than or equal to high.

The printArray function prints the elements of the array in a single line.

In main, the program initializes the array, prints the unsorted array, sorts it using quicksort, and then prints the sorted array. It also measures the time before and after calling the quicksort function.

Section 3: Output



```
Last login: Wed Dec  4 14:53:19 on ttys001
/Users/qaasidbajwa/Desktop/quicksort ; exit;
qaasidbajwa@Qaasids-MacBook-Pro ~ % /Users/qaasidbajwa/Desktop/quicksort ; exit;

Unsorted Array:
8 7 2 1 0 9 6 3 5 4
Sorted Array in Ascending Order:
0 1 2 3 4 5 6 7 8 9
Time taken by Quicksort: 9 microseconds
```

This is my output. We can see the unsorted and sorted arrays. The time taken by Quicksort was 9 microseconds.

Section 4: Manual Quicksort Example

Example Array
[8,3,7,5,2]

Step 1: Choose Pivot
my pivot will be 2

Step 2: Partition Array

Left Part ≤ to pivot
Right Part > than pivot
Start with two pointers
i (boundary of smaller elements): initialize to -1
j (current element): Move from start to second to last element

Compare each element with pivot
≤ pivot, increment i & swap array[i] w/ array [j]
> pivot, do nothing

j=0 8>2 no swap
j=1 3>2 no swap
j=2 7>2 no swap
j=3 5>2 no swap
swap pivot with array[i+1]
$\Rightarrow$ [2,3,7,5,8]   Pivot Position: 0
Step 3: Recursively Sort Subarrays
Left Subarray: no elements []
Right Subarray: [3,7,5,8] Repeat Process
[3,7,5,8] Pivot: 8
Partition   Compare
3≤8 swap with itself
7≤8 swap w/ itself
5≤8 swap w/ itself
$\Rightarrow$ [3,7,5,8]   Pivot Position: 3

Left Subarray of [3,7,5]: [3,7,5]   Pivot: 5
Partition: Compare
3≤5 swap with itself
7>5 no swap

Resulting Subarray: [3,5,7] Pivot Position: 1

Final subarrays

Left [3] Already sorted

right [7]

Combine all subarrays

[2,3,5,7,8]