Implementing Sorting Algorithms in C and Creating a Shared Library for Python

Yousha Munowar

McMaster University

MECHTRON 2MP3

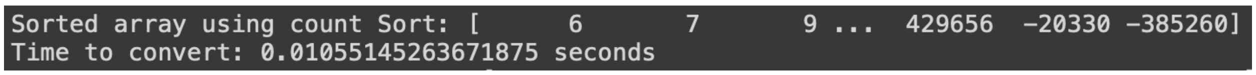
Pedram Pasandide

October 14th 2024

Time Complexity chart

Sort Method	**Complexity**		**CPU Time (sec)**
	Time	**Space**	
Bubble	$O(n^2)$	$O(1)$	662.35242
Insertion	$O(n^2)$	$O(1)$	58.1352
Merge	$O(n\ \log(n))$	$O(n)$	0.078293
Heap	$O(n\ \log(n))$	$O(1)$	0.10074
Counting	$O(n\ + k)$	$O(k)$	0.01055
Built-in sorted()	$O(n\ \log(n))$	$O(n)$	0.57202
numpy.sort()	$O(n\ \log(n))$	$O(n)$	0.04884

Although count sort can be seen to be the fastest sorting method, however when analyzing the count sort output result, we can see it does not completely sort the array. This is due to the method's lack of ability to sort negative numbers in an array. This is rooted from the problem of the way it arranges the array, thus needing all the integers within the array to be greater than 0. This can be see within the result output of the counting sorting method.



```
Sorted array using count Sort: [      6       7       9 ...  429656  -20330 -385260]
Time to convert: 0.01055145263671875 seconds
```

To get a more reliable and accurate result, we can change the range of numbers to exclude any negative integer values. However, we won't change the size of range of numbers, or in other words, we initially had a range of (-1000000,1000000) with a total of two-million unique integers, thus our new range will be (0,2000000), giving us the same number of unique integers. This change gives us the following output of results:

Sort Method	Complexity		CPU Time (sec)
	Time	Space	
Bubble	$O(n^2)$	$O(1)$	670.23281
Insertion	$O(n^2)$	$O(1)$	59.31016
Merge	$O(n \log(n))$	$O(n\ )$	0.08258
Heap	$O(n \log(n))$	$O(1)$	0.10169
Counting	$O(n + k)$	$O(k)$	0.01934
Built-in sorted()	$O(n \log(n))$	$O(n\ )$	0.49102
numpy.sort()	$O(n \log(n))$	$O(n\ )$	0.05031

.

Algorithms

Throughout this test case, we used 7 different types of sorting methods to compare the times of each and see how they perform with large test cases. Firstly, we tested Bubble sort, the most inefficient out of the methods and utilizes the idea of swapping elements if they are in the wrong order to sort the array. We also tested Insertion sort, although much more efficient than bubble sort, it is much less efficient compared to the others, this method builds a new sorted array one element at a time putting them in the correct position. Merge sort was another method which came to be very efficient that divide and conquers an array and solving smaller bits at a time. With a similar time performance, Heap sort is another efficient method that uses binary heap data structures to create a sorted array. The last sorting method was counting sort which proved to be extremely efficient but not able to do negative values, which uses a comparison-based algorithm efficient for a range of integers. As for the last two, one uses python's built-in sorting method, and the other uses NumPy's optimized sorting method which is the most optimized.

Compiling and Running

Due to the various complications of compiling this program as a saved library with my MacBook Air M2 chip, I had taken a different approach of compiling the code outside of my terminal.

1. Upload both the **mySort_test.ipynb** file and **mySort.c** file onto your google drive
2. Open the **mySort_test.ipynb** file within google colab and run each cell
3. One of the cells will utilize google colab's built-in shell terminal to compile the file as a shared library.
4. Run the file as normal.

Because of me using the built-in shell terminal, there is no reason for me to create a Makefile.

Appendix

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *x, int *y) {
int temp = *x;
*x = *y;
*y = temp;
}

// Bubble Sort
void bubbleSort(int arr[], int n) {
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (arr[j] > arr[j + 1])
swap(&arr[j], &arr[j + 1]);
}
}
}

//Insertion Sort

void insertionSort(int arr[], int n)
{
for (int i = 1; i < n; ++i) {
int key = arr[i];
int j = i - 1;

while (j >= 0 && arr[j] > key) {
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
}


//Merge sort


void merge(int arr[], int left, int mid, int right) {
int n1 = mid - left + 1;
int n2 = right - mid;
int L[n1],R[n2];

for (int i = 0; i < n1; i++)
L[i] = arr[left + i];
for (int j = 0; j < n2; j++)
R[j] = arr[mid + 1 + j];

int i = 0;
int j = 0;
int k = left;
while (i < n1 && j < n2) {
if (L[i] <= R[j]) {
arr[k] = L[i];
i++;
} else {
arr[k] = R[j];
j++;
}
k++;
}

while (i < n1) {
arr[k] = L[i];
i++;
k++;
}

while (j < n2) {
arr[k] = R[j];
j++;
k++;
}

}

void mergeSort(int arr[], int left, int right) {
if (left < right) {
int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);
```

```
}
}



//Heap Sort

void heapify(int arr[], int n, int i) {
int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;
if (left < n && arr[left] > arr[largest])
largest = left;

if (right < n && arr[right] > arr[largest])
largest = right;

if (largest != i) {
int temp = arr[i];
arr[i] = arr[largest];
arr[largest] = temp;

heapify(arr, n, largest);
}
}

void heapSort(int arr[], int n) {
for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {
int temp = arr[0];
arr[0] = arr[i];
arr[i] = temp;

heapify(arr, i, 0);
}
}

//Counting Sort

void countSort(int arr[], int n) {
int max = arr[0];
for (int i = 1; i < n; i++) {
```

```
if (arr[i] > max)
max = arr[i];
}

int* count = (int *)calloc(max + 1, sizeof(int));

for (int i = 0; i < n; i++) {
count[arr[i]]++;
}

int index = 0;
for (int i = 0; i <= max; i++) {
while (count[i] > 0) {
arr[index] = i;
index++;
count[i]--;
}
}
}


int main() {
return 0;
}
```

References

Josh Morrison 7. (1956, June 1). *What is Newton-Raphson Square Method's time complexity* Stack Overflow. https://stackoverflow.com/questions/5005753/what-is-newton-raphson-square-methods-time-complexity

GeeksforGeeks. (2024, June 25). *Sqrt() function in C*. https://www.geeksforgeeks.org/sqrt-function-in-c/

Chatgpt. (2022, November 30). https://chatgpt.com/

For additional information on APA Style formatting, please consult the APA Style Manual, 7th Edition.