**Lab 5**

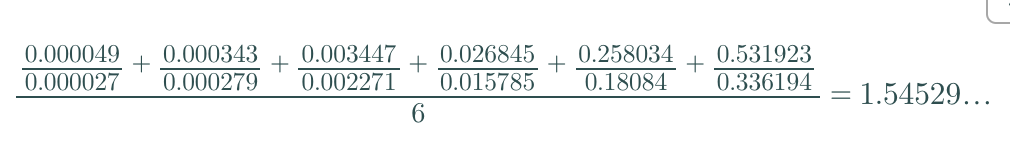
**Isa Dzhumabaev**

**1) Compare randomized quicksort to quicksort and quadsort using various input scenarios.**

I used Quick Sort algorithm from previous labs and added **rand\_quickSort()** function that uses randomized pivot. First, my program generates an array with N random integer numbers, then sends it to standard **norm\_quickSort(**) and calculates its time, after that it sends the same array to randomized **rand\_quickSort()** function and also calculates its time.

As you can see in screenshot below randomized quick sort was faster in all cases.

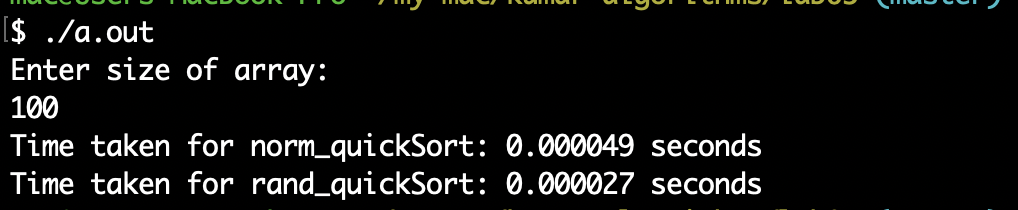
I used some simple formula to calculate average ratio and we can see that randomized quick sort is about 1.5 times faster then standard quick sort.



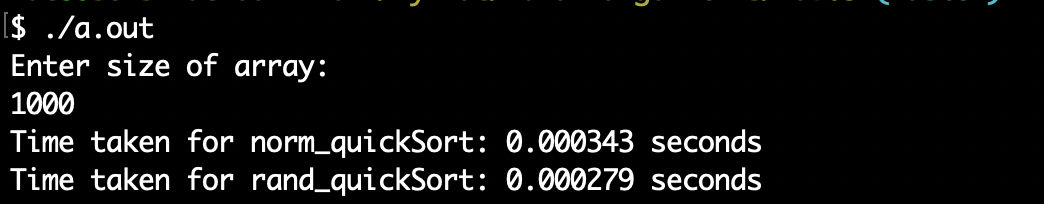
**2) Record the results.**

**Screenshots of comparisons:**

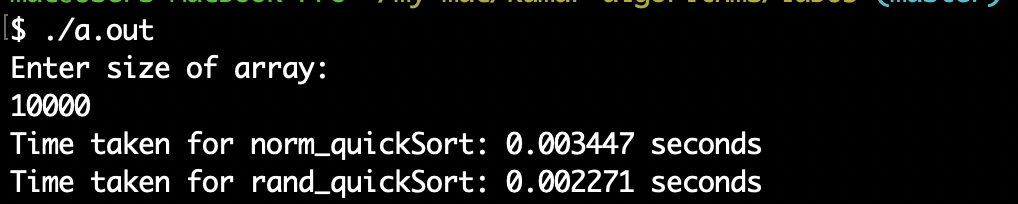
N = 100



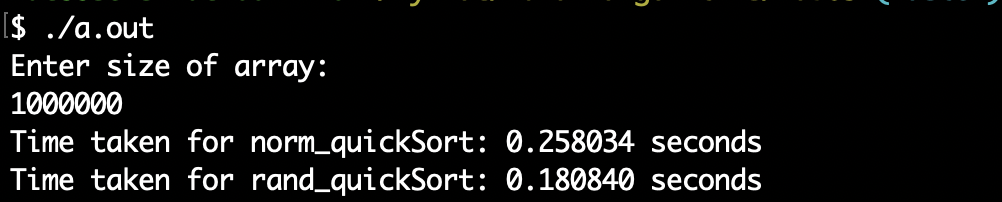
N = 1000



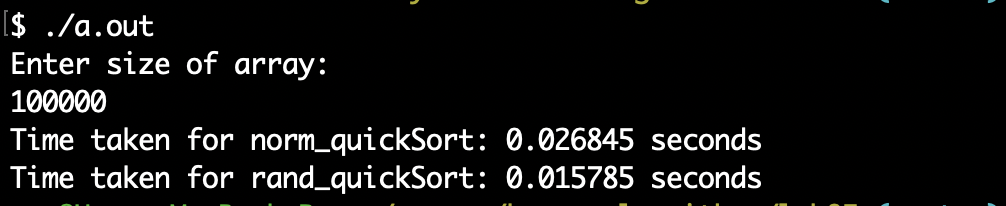
N = 10 000



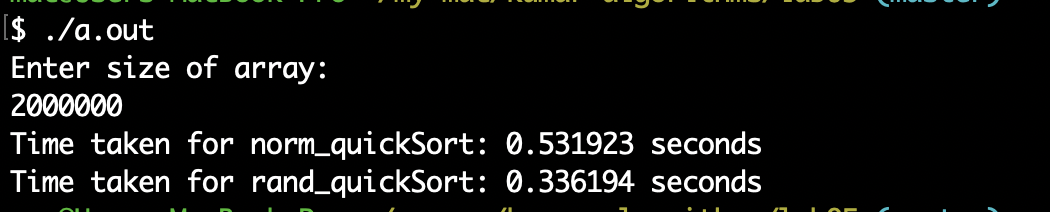
N = 100 000



N = 1 000 000



N = 2 000 000



**3) Which one is better and why?**

Randomized quick sort is better as it lower chances that after partition operation we get one array of n-1 elements and second with 0 elements. This situation is the worst case and leads to O(n\*n) in case we get such a partition in every partition operation. By lowering chances for worst case we increase chances for O(n\*log(n)) time which is obviously a better option.

**Source code:**

#include "stdio.h"

#include "time.h"

#include "stdlib.h"

int partition (int\* arr, int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

**for** (int j = low; j < high; j++)

{

**if** (arr[j] < pivot)

{

i++;

int t = arr[i];

arr[i] = arr[j];

arr[j] = t;

}

}

int t = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = t;

**return** (i + 1);

}

int partition\_r(int\* arr, int low, int high)

{

srand(time(NULL));

int random = low + rand() % (high - low);

int t = arr[random];

arr[random] = arr[high];

arr[high] = t;

**return** partition(arr, low, high);

}

void quickSort(int\* arr, int low, int high)

{

**if** (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void rand\_quickSort(int\* arr, int low, int high)

{

**if** (low < high)

{

int pi = partition\_r(arr, low, high);

rand\_quickSort(arr, low, pi - 1);

rand\_quickSort(arr, pi + 1, high);

}

}

int main(int argc, char\*\* argv) {

int sz = -1;

printf("Enter size of array:**\n**");

scanf("%d", &sz);

int arr[sz];

srand(time(NULL));

**for** (int i = 0; i < sz; ++i)

{

arr[i] = rand() % (sz \* 20);

}

clock\_t begin = clock();

rand\_quickSort(arr, 0, sz - 1);

clock\_t end = clock();

printf("Time taken for norm\_quickSort: %f seconds**\n**", ((double) (end - begin)) / CLOCKS\_PER\_SEC);

begin = clock();

rand\_quickSort(arr, 0, sz - 1);

end = clock();

printf("Time taken for rand\_quickSort: %f seconds**\n**", ((double) (end - begin)) / CLOCKS\_PER\_SEC);

**return** 0;

}