VE281 C1

517370910104 Shuyi Zhou

May 2020

1 Introduction

In this assignment, I will compare the performance for six sorting algorithm: bubble sort(0), inserting sort(1), selection sort(2), merge sort(3), quick sort with extra array(4) and quick sort with in place partition(5).

2 Time complexity via different sizes

I tested arrays with size 10, 1000, 10000, 20000, 40000, 80000. Every size I tested 20 different arrays and got their average. For fair comparisons, the same set of arrays were applied to all the algorithms. The detailed tables can be checked in Appendix A, while the average valued table is the following (Table 1):

No:Size	10	100	1000	10000	20000	40000	80000
0	2.2	29.85	2311.9	302534.15	1271278.3	5392457.2	21336918.85
1	1.55	8.35	636.35	62988.1	247108.8	1014674.9	3943467.15
2	1.55	15.7	1195.75	118028.95	436065.05	1751259.2	6731936.75
3	2.25	11	119.95	1394	2993.7	6556.05	13672.65
4	12.55	66.35	654.75	7155.5	13765.85	28319.9	54809.05
5	12.5	64.6	645.35	6682.75	13162.2	27176.55	52516.55

Table 1: sort method with average time (ticks). No. : bubble sort(0), inserting sort(1), selection sort(2), merge sort(3), quick sort with extra array(4) and quick sort with in place partition(5)

3 Analysis

I plotted the result into a graph (Figure 1). In the figure, I also put in n^2 and $nlogn$ via size n, which are dark blue and brown lines respectively. Also, in the Figure, quick sort with in place partition and quick sort with extra array partition are almost the same line, so we can hardly distinguish them.
So what we can find in graph is that after the size is larger than 100, the trends of bubble sort, insertion sort and selection sort are almost the same as that of

n^2, while the trends of merge sort and quick sort are almost the same as that of $nlogn$.
So, as said in lectures, the average time complexity of bubble sort, insertion sort and selection sort is $O(size)$, while the average time complexity of quick sort and merge sort is $O(logn)$.

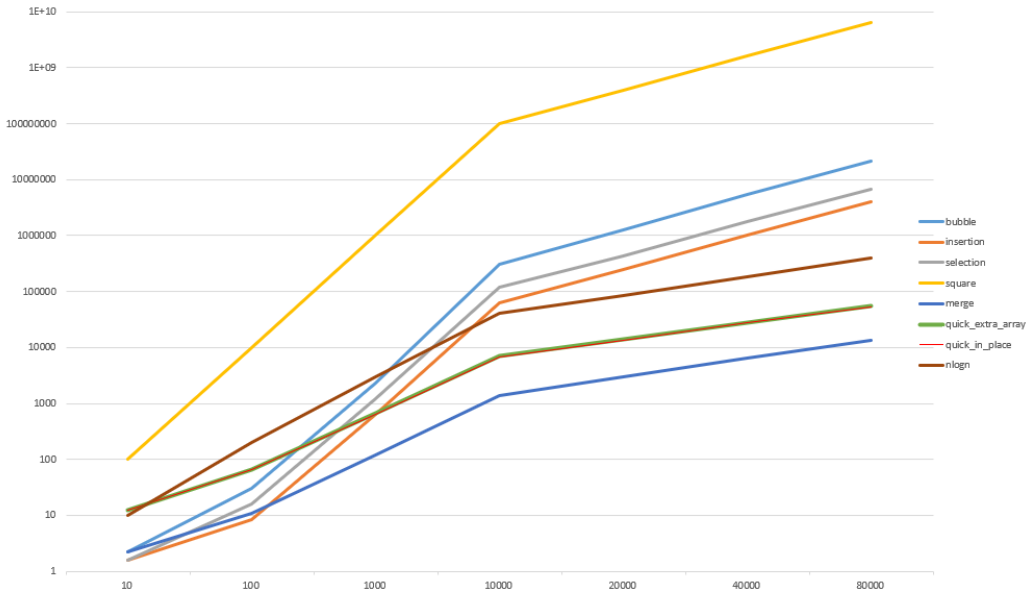

Figure 1: size, algorithm via ticks the algorithm consumed.

4 Discussion

I plot a second graph which focused on the two quick sorts: Figure 2. In this figure, I also plotted n. I found n and $nlogn$ have very similar trends and it is hard to distinguish whether the time complexity of the sort algorithms is $O(n)$ or $O(nlogn)$ exactly. I think if we can have larger size, the trend might be more clear. However, it takes too long time to run it on my computer.

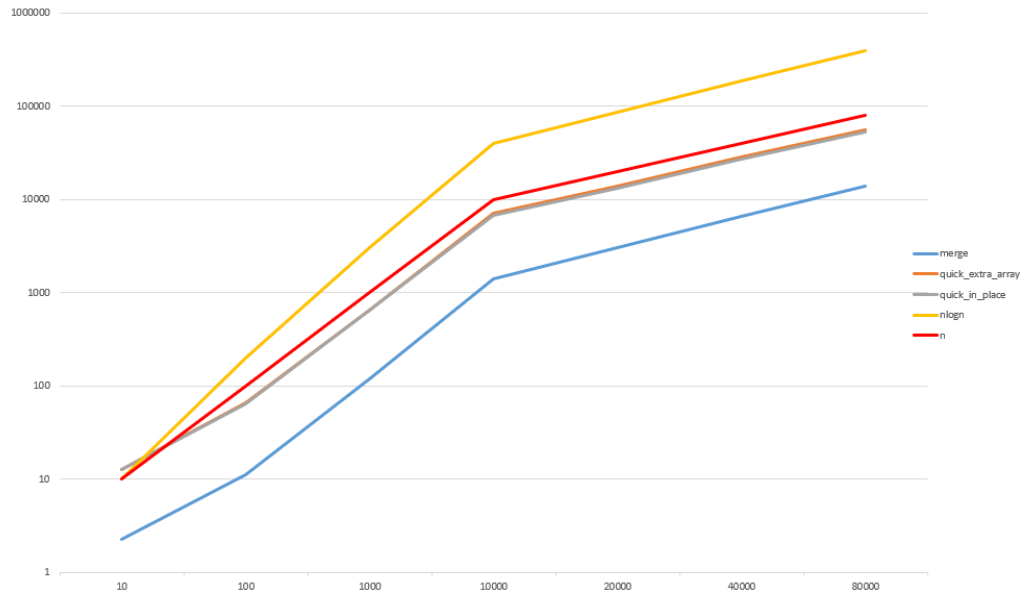

Figure 2: Meger sort and quick sort and nlogn and n via ticks they consume.

A Tables for every size test

Sort No.	1	2	3	4	5	6	7	8	9	10	
0	5	5	6	3	2	2	1	1	2	2	
1	4	3	4	2	2	1	1	1	1	1	
2	4	3	3	2	2	1	1	1	1	1	
3	6	5	4	2	2	2	2	2	1	2	
4	28	29	18	13	13	10	9	12	10	11	
5	33	30	17	11	12	10	9	11	9	9	
	11	12	13	14	15	16	17	18	19	20	Average
0	1	1	2	1	1	2	2	2	2	1	2.2
1	1	1	1	1	1	1	1	1	2	1	1.55
2	1	1	1	1	2	1	1	1	1	2	1.55
3	2	2	1	2	1	2	2	2	1	2	2.25
4	9	10	10	11	10	10	10	9	9	10	12.55
5	11	9	9	10	9	10	9	11	10	11	12.5

Table 2: tests for size 10.

Sort No.	1	2	3	4	5	6	7	8	9	10	
0	29	51	30	28	29	27	28	31	28	29	
1	8	8	9	9	8	9	8	9	8	8	
2	16	16	16	16	16	17	15	15	15	16	
3	10	10	11	10	10	10	10	10	10	12	
4	68	66	66	66	64	64	67	64	65	64	
5	64	79	64	77	61	62	61	73	62	62	
	11	12	13	14	15	16	17	18	19	20	Average
0	28	29	29	29	30	29	29	28	28	28	29.85
1	9	8	9	9	8	8	8	8	8	8	8.35
2	15	15	15	16	16	16	16	16	15	16	15.7
3	9	27	10	10	10	10	10	11	10	10	11
4	65	67	64	66	66	67	66	64	65	83	66.35
5	62	62	61	62	64	64	64	64	61	63	64.6

Table 3: tests for size 100.

sort No.	1	2	3	4	5	6	7	8	9	10	
0	2300	2293	2224	2496	2216	2344	2269	2221	2401	2332	
1	623	626	608	766	627	613	609	577	576	611	
2	1168	1167	1139	1219	1223	1211	1138	1112	1078	1138	
3	115	115	110	110	113	111	105	112	106	113	
4	641	640	623	699	789	624	796	627	772	628	
5	629	680	686	613	611	610	581	688	613	830	
	11	12	13	14	15	16	17	18	19	20	Average
0	2400	2304	2295	2288	2297	2228	2244	2228	2632	2226	2311.9
1	631	574	611	676	641	683	778	668	607	622	636.35
2	1142	1579	1142	1441	1146	1142	1146	1214	1148	1222	1195.75
3	133	129	179	111	111	111	188	105	111	111	119.95
4	593	591	624	627	629	594	626	626	723	623	654.75
5	617	579	680	678	681	616	612	677	614	612	645.35

Table 4: tests for size 1000.

sort No.	1	2	3	4	5	
0	281590	281437	299455	306294	292253	
1	59860	59157	62421	61332	59673	
2	111071	110605	114477	120864	119587	
3	1330	1309	1391	1312	1304	
4	6680	6869	6680	6284	6578	
5	6039	6268	6497	6642	6621	
	6	7	8	9	10	
0	295889	291882	294147	312125	310399	
1	63846	62633	64705	62803	64193	
2	117886	115125	119186	121878	121778	
3	1305	1308	1474	1383	1446	
4	6812	6745	6925	7444	6737	
5	6005	6038	7005	6932	6969	
	11	12	13	14	15	
0	293871	306569	320947	327177	311292	
1	62874	64056	64445	65842	65872	
2	116180	120753	115221	123653	121298	
3	1377	1444	1335	1493	1427	
4	7008	7581	7999	7638	7875	
5	6627	7180	6816	7048	6818	
	16	17	18	19	20	Average
0	300933	302160	289306	294971	337986	302534.15
1	65400	58432	63180	63641	65397	62988.1
2	118170	112029	115845	119166	125807	118028.95
3	1409	1448	1516	1418	1451	1394
4	7237	6825	7788	8103	7302	7155.5
5	6432	6113	6377	6631	8597	6682.75

Table 5: tests for size 10000.

sort No.	1	2	3	4	5	
0	1337880	1323611	1277530	1260085	1246638	
1	259553	259121	270766	242075	249019	
2	444117	460721	455877	445048	425530	
3	2833	3121	3234	3223	3052	
4	13280	13582	13594	13671	13705	
5	13063	12902	14425	12701	12760	
	6	7	8	9	10	
0	1296049	1301731	1237595	1220034	1264043	
1	254225	235743	237646	231762	247816	
2	440883	424835	445000	436681	439029	
3	3100	2844	2789	3089	3123	
4	13955	14151	13436	14010	14004	
5	13412	12719	12513	13271	12965	
	11	12	13	14	15	
0	1269817	1260571	1260789	1266649	1265584	
1	248097	247154	248064	249683	249494	
2	433498	430139	431978	431741	429638	
3	3033	3001	2973	3032	3036	
4	14038	14138	14439	13742	13850	
5	13290	13512	14037	13544	13428	
	16	17	18	19	20	Average
0	1275708	1267224	1259088	1264724	1270216	1271278.3
1	235023	247374	248467	246963	234131	247108.8
2	420977	430536	431666	417317	446090	436065.05
3	2817	2844	3036	2851	2843	2993.7
4	13230	13723	14350	13492	12927	13765.85
5	12813	13039	13383	12955	12512	13162.2

Table 6: tests for size 20000.

sort No.	1	2	3	4	5	
0	5351744	5155300	5529878	5495935	5500780	
1	1036547	1020441	1038774	1034388	1035440	
2	1814923	1810533	1777034	1773662	1774099	
3	7185	6058	6876	6805	6761	
4	30170	28156	28662	28931	28739	
5	27992	26991	27804	27538	27462	
	6	7	8	9	10	
0	5274623	5555234	5489713	5499163	5681419	
1	1024790	1041904	1035665	1030006	1026708	
2	1774557	1775397	1772767	1781864	1768891	
3	6821	6782	6831	6597	6763	
4	28749	28832	29090	28712	28553	
5	27666	27458	27645	27516	27385	
	11	12	13	14	15	
0	5480844	5481425	5470828	5460276	5203139	
1	1025983	1024209	976347	973241	984638	
2	1763529	1770379	1712526	1688602	1682095	
3	6832	6838	5953	6407	6501	
4	28794	28693	26014	27841	28017	
5	27534	27375	26087	27470	25784	
	16	17	18	19	20	Average
0	4931653	5236206	5446865	5395367	5208752	5392457.2
1	979795	990039	1014460	1014652	985471	1014674.9
2	1688803	1736161	1683379	1743611	1732372	1751259.2
3	6427	6042	6466	6235	5941	6556.05
4	28169	26492	28328	27664	27792	28319.9
5	26900	26311	26305	28627	25681	27176.55

Table 7: tests for size 40000.

sort No.	1	2	3	4	5	
0	22392763	22659220	20729704	20463664	21408541	
1	4161646	4132798	4066801	3960775	3927590	
2	7187627	6771296	7100880	6739382	6629837	
3	15504	13796	13429	13824	13963	
4	63851	52932	53448	55838	55822	
5	51339	53002	51215	53690	53177	
	6	7	8	9	10	
0	21533806	20852309	21171179	20958523	21489070	
1	3906652	3865036	3854488	3890529	3916849	
2	6569732	6807991	6575864	6579409	6617501	
3	12945	13489	13750	13640	13765	
4	53665	52480	53967	54972	55087	
5	51976	51045	53036	53030	53190	
	11	12	13	14	15	
0	21179828	22006008	20572568	21831668	21712097	
1	3912884	3894465	3927255	4032025	3855082	
2	6617300	7018224	6889091	7018856	6539266	
3	13029	13913	13342	13767	12794	
4	53394	57001	55798	53130	53374	
5	52510	54502	51499	52358	52855	
	16	17	18	19	20	Average
0	21228938	20922410	20952960	21129970	21543151	21336918.85
1	3877855	3842396	3949668	3956593	3937956	3943467.15
2	6605339	6526384	6596967	6738671	6509118	6731936.75
3	13359	13259	14613	14385	12887	13672.65
4	55320	54166	54465	53635	53836	54809.05
5	52734	52596	51185	52083	53309	52516.55

Table 8: tests for size 80000.

B main.cpp

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


// Swap function
void
swap(int* array, int index1, int index2)
{
    int temp = array[index1];
    array[index1] = array[index2];
    array[index2] = temp;
}

// Bubble sort (0)
void
bubble(int* array, int arraySize)
{
    // Swapping.
    for (int i = arraySize - 2; i >= 0; i--)
    {
        for (int j = 0; j <= i; j++)
        {
            if (array[j] > array[j+1])
            {
                swap(array, j+1, j);
            }
        }
    }
}

// Insertion sort (1)
void
insertion(int* array, int arraySize)
{
    for (int i = 1; i < arraySize; i ++)
    {
        int temp = array[i];
        // shift sorted elements greater than t right, and then insert
            temp in gap.
        int j;
        for (j = i - 1; j >= 0; j--)
        {
            if (array[j] > temp) array[j+1] = array[j];
            else break;
        }
```

```
        array[j+1] = temp;
    }
}

// Selection sort (2)
void selection(int* array, int arraySize)
{
    for (int i = 0; i < arraySize - 1 ; i ++)
    {
        int min_value = array[i];
        int min_index = i;
        int temp = array[i];
        for (int j = i + 1; j <= arraySize - 1; j++){
            if (array[j] < min_value)
            {
                min_value = array[j];
                min_index = j;
            }
        }
        array[i] = min_value;
        array[min_index] = temp;
    }
}

// Merge sort (3)
void
merge (int* array, int left, int mid, int right)
{
    int mergeArray[right - left + 1];
    int i = left;
    int j = mid + 1;
    int k = 0;
    while (i < mid + 1 && j <= right) {
        if(array[i] <= array[j]) mergeArray[k++] = array[i++];
        else mergeArray[k++] = array[j++];
    }
    if (j > right)
    {
        while(i<mid+1) mergeArray[k++] = array[i++];
    } else
    {
        while(j<=right) mergeArray[k++] =array[j++];
    }
    for(int n = left; n<=right; n++) array[n] = mergeArray[n-left];
}

void
mergesort(int* array, int left, int right)
{
    if (left >= right) return;
```

```
    int mid = (left + right) / 2;
    mergesort(array, left, mid);
    mergesort(array, mid+1, right);
    merge(array, left, mid, right);
}


// Quick sort using extra array (4)
int
partition_extra_array (int* array, int left, int right)
{
    srand(time(NULL));
    int temp = rand() % (right - left + 1);
    int array_temp[right - left + 1];
    int pivot = array[temp + left];
    swap(array, left, temp + left);
    int i_left = 0, i_right = right - left;
    for (int i = left + 1; i <= right; i++)
    {
        if (array[i] < pivot) array_temp[i_left ++] = array[i];
        else array_temp[i_right--] = array[i];
    }
    array_temp[i_left] = pivot;
    for (int i = 0; i<=right - left; i++) array[left + i] =
         array_temp[i];
    return i_left + left;
}

void
quicksort_extra_array (int* array, int left, int right)
{
    int pivotat;
    if (left >= right) return;
    pivotat = partition_extra_array (array, left, right);
    quicksort_extra_array(array, left, pivotat-1);
    quicksort_extra_array(array, pivotat+1, right);

}

// Quick sort with in-place partitioning (5)
int
partition_in_place(int* array, int left, int right) {
    srand(time(NULL));
    int temp = rand() % (right - left + 1);
    int pivot = array[temp + left];
    swap(array, left, temp+left);
    int i = left + 1, j = right;
    while (true)
    {
        while(array[i] < pivot && i <= right) i++;
```

```
        while(array[j] >= pivot && j >= left) j--;
        if(i<j)
        {
            // Swap
            swap(array, i, j);
        } else break;
    }
    if(j<left) j = left;
    // Swap the first with A[j]
    swap(array, left, j);

    return j;
}


void
quicksort_in_place (int* array, int left, int right)
{
    int pivotat;
    if (left >= right) return;
    pivotat = partition_in_place (array, left, right);
    quicksort_in_place(array, left, pivotat-1);
    quicksort_in_place(array, pivotat+1, right);
}

int
main()
{
    int whichSort;
    int arraySize;

    // Read file.
    cin>>whichSort;
    cin>>arraySize;
    // Define Array.
    int* array = new int[arraySize];
    for(int i = 0; i < arraySize ; i++) cin>>array[i];
    // Sort
   clock_t t = clock();
    switch (whichSort) {
        case 0:
            bubble(array, arraySize);
            break;
        case 1:
            insertion(array, arraySize);
            break;
        case 2:
            selection(array, arraySize);
            break;
        case 3:
```

```
            mergesort(array, 0, arraySize-1);
            break;
        case 4:
            quicksort_extra_array(array, 0, arraySize - 1);
            break;
        default:quicksort_in_place(array, 0, arraySize - 1);

    }
    // for test time, de-comment this line
// cout << clock() - t<< endl;
    // Save to output.
    // when testing time, comment this line
    for (int i = 0; i < arraySize; i++) cout<<array[i]<<endl;
    // Release memory.
    delete [] array;
    return 0;
}
```

C generate$_{a}rray.cpp$

```
#include <iostream>
#include <ctime>
#include <fstream>

using namespace std;

int main(int argc, char **argv){
    ofstream file0;
    ofstream file1;
    ofstream file2;
    ofstream file3;
    ofstream file4;
    ofstream file5;
    ofstream file6;

    file0.open("./array0");
    file1.open("./array1");
    file2.open("./array2");
    file3.open("./array3");
    file4.open("./array4");
    file5.open("./array5");

    file0<<"0\n";
    file1<<"1\n";
    file2<<"2\n";
    file3<<"3\n";
    file4<<"4\n";
```

```
    file5<<"5\n";

    srand(time(NULL));
   string size_ = argv[1];
    int size = stoi(size_);


    file0<<size;
    file1<<size;
    file2<<size;
    file3<<size;
    file4<<size;
    file5<<size;
    file0<<'\n';
    file1<<'\n';
    file2<<'\n';
    file3<<'\n';
    file4<<'\n';
    file5<<'\n';

    for(int i = 0; i<size; i++)
    {
        int temp = mrand48();
        file0<<temp;
        file1<<temp;
        file2<<temp;
        file3<<temp;
        file4<<temp;
        file5<<temp;
        file0<<'\n';
        file1<<'\n';
        file2<<'\n';
        file3<<'\n';
        file4<<'\n';
        file5<<'\n';
    }

    file1.close();
    file2.close();
    file3.close();
    file4.close();
    file5.close();
    file0.close();
    return 0;
}
```

D Makefile

```
all: main clean

main: main.o
   g++ -o main main.cpp

main.o: main.cpp
   g++ -c main.cpp

generate: generate_array.cpp
   g++ -o generate generate_array.cpp

clean:
   rm -f *.o
```

E shell: get time

```
for i in 10 100 1000 10000 20000 40000 80000
do
   echo "now size $i"
   for n in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
   do
      ./generate $i
      echo "$n :"
      ./main<array0
      ./main<array1
      ./main<array2
      ./main<array3
      ./main<array4
      ./main<array5
   done
done
```