Problem Set #2 (Algorithms)

Department: 컴퓨터정보공학부

Student ID: 2015722025

Student Name: 정용훈

1. **To show the results and graphs in Problem 2, 3, and 4, write your program with your comments. Explain your program at least four lines.**

```
#include<iostream>
#include<stdlib.h>
#include<time.h>
#include<windows.h>
#include<chrono>
#include<cmath>
using namespace std;

void Create_array(int A[], int size);
void textcolor(int foreground, int background);
int Partition(int Array[], int p, int q);
void Quicksort(int Array[], int p, int r,int option);
void Problem2();
void Problem3();
void Problem4();
void FillArray(int Array[], int size);
void swap(int Array[], int i, int j);
void CreateBestcase(int Array[], int begin, int end);
void CreateRandomcase(int Array[], int size);
void InitializeArray(int Array[], int size);
void printArray(int Array[], int p, int r, int size);
int sizeArray(int Array[]);

#define LIGHTRED 12
#define LIGHTGREEN 10
#define DARKGRAY 8
#define YELLOW 14
#define WHITE 15
#define BLACK 0
#define GREEN 12
#define NOP -100

int Info_count = 0;
int Info_count2 = 0;
int* Info_Worst;
int* Info_Best;
int* Info_Average;
int P_flag = 0;
```

해당 코드는 전방선언과, 각종 출력을 위한 정의 선언과 전역변수들이다. 전방 선언된 함수들을 보면, quick sort와 관련된 swap, 배열을 생성하는 함수, 출력하는 함수와 문제로 제시 받은 문항을 해결하기 위한 Problem2, 3, 4 함수들로 이루어진 것을 확인할 수 있다. Problem 함수를 실행하면 과제로 나온 문제를 각종 전방함수와, 전역변수를 사용하여 동작, 출력 해준다.

```
int main()
{
    int button = 0; //For checking select menu
    cout << "*************************" << endl;
    cout << "*      Start Program      *" << endl;
    cout << "*       Algorithms        *" << endl;
    cout << "*       2015722025        *" << endl;
    cout << "*     Jeong yong hoon     *" << endl;
    cout << "*************************" << endl;

    while (1)
    {
        cout << endl;
        cout << "1.Problem 2" << endl; //menu of view
        cout << "2.Problem 3" << endl;
        cout << "3.Problem 4" << endl;
        cout << "4.Exit Program" << endl;
        cout << "Select menu: ";
        cin >> button;

        switch (button)
        {
        case 1:
            Problem2(); //Function of problem 2
            break;
        case 2:
            Problem3(); //Function of problem 3
            break;
        case 3:
            Problem4(); //function of problem 4
            break;
```

```
        case 4:
            cout << "*************************" << endl; //end of program
            cout << "*      End of Program     *" << endl;
            cout << "*       Algorithms        *" << endl;
            cout << "*       2015722025        *" << endl;
            cout << "*     Jeong yong hoon     *" << endl;
            cout << "*************************" << endl;
            return 0;
        }
    }
}
```

다음은 main함수로 문제에 대한 동작을 사용자가 선택하여 출력할 수 있도록 되어있으며, 문제 3개에 대한 선택과 프로그램을 종료할 수 있는 선택지를 만들어 두었다.

```
void textcolor(int foreground, int background) //For change text color of console
{
    int color = foreground + background * 16;
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), color);
}
```

시각적인 부분을 위해 콘솔에서 문자의 색을 변환시켜주는 code이다.

```
89  void Create_array(int A[], int size) //For creating random array
90  {
91      int item;
92      srand(time(NULL));
93      for (int i = 0; i < size; i++)
94      {
95      again: //lable
96          item = rand() % size + 1;
97
98          for (int j = 0; j < size; j++)
99              if (item == A[j]) //check items
00                  goto again;
01          A[i] = item;
02      }
03  }
```

Average case측정을 위한 Array생성 함수로써, 중복되는 수가 없도록 중복 check기능이 포함된 함수이다.

```
105 int Partition(int Array[], int p, int q) //The function of partition of qucik sort
106 {
107     int x = Array[p]; //pivot
108     int i = p;
109     int tmp = 0;
110
111     for (int j = p + 1; j <= q; j++)
112     {
113         Info_count++; //For checking comparison
114         if (Array[j] <= x)
115         {
116             i++;
117             tmp = Array[i];
118             Array[i] = Array[j];
119             Array[j] = tmp;
120         }
121     }
122     tmp = Array[p]; //swap
123     Array[p] = Array[i];
124     Array[i] = tmp;
125
126     return i; //return i
127 }
```

교재와 강의자료를 토대로 작성한 Quick sort의 Partition함수이며, 다음 Sub Array가 나뉘는 기준을 return해주는 역할을 한다.

```
129 int Random_Partition(int Array[], int p, int q, int option)  //The function of partition
130 //of randomized qucik sort
131 {
132     int x = (rand() % (q - p + 1)) + p; //select random pivot
133 
134     if (option == 1)
135     {
136         cout << "Select    : ";
137         printArray(Array, x, NOP, sizeArray(Array));
138     }
139 
140     int tmp = Array[x]; //move pivot is selected
141     Array[x] = Array[p];
142     Array[p] = tmp;
143 
144     if (option == 1)
145     {
146         cout << "Swap pivot: ";
147         printArray(Array, p, q, sizeArray(Array));
148     }
149     return Partition(Array, p, q); //practice normal partition
150 }
```

다음은 Randomized quick sort를 위한 partition이다. 기존에 pivot을 가장 처음 item에서 추출하였다면, pivot을 랜덤으로 선택하여, 성능을 개선한 알고리즘이다. 위 code를 보면 랜덤으로 pivot을 선택하고, 가장 앞 item과 바꾼 후 기존 partition을 통해 값을 return해 주는 것을 확인할 수 있다. 결론적으로 Random으로 pivot을 선택하고 선택된 pivot을 가장 앞으로 바꿔줬다는 것만 다르다.

```
152 void Quicksort(int Array[], int p, int r, int option) //Function of Quick sort
153 {
154     if (p < r) //check number of items
155     {
156         if(option==1)
157             printArray(Array, p, r, sizeArray(Array));
158 
159         int q = Partition(Array, p, r); //practice partition
160         Quicksort(Array, p,q-1,option); //recursive function
161         Quicksort(Array, q + 1, r,option);
162     }
163 }
```

Quick sort함수이다. 교수님의 강의와 교재에서 배운 것에 동일하게 partition을 통해 subarray를 나눠주고 각각의 subarray를 재귀 함수를 통하여 quick sort를 반복 실행하는 모습을 확인할 수 있다.

```
165 void Random_Quicksort(int Array[], int p, int r, int option) //Randomized quick sort
166 {
167     if (p < r) //Remaining content is equal to normal Quick sort Exclude partition function
168     {
169         int q = Random_Partition(Array, p, r, option);
170         Random_Quicksort(Array, p, q - 1, option);
171         Random_Quicksort(Array, q + 1, r, option);
172     }
173 }
```

마찬가지로 Randomized quick sort를 위한 함수이며, 이는 partition의 동작만 약간 다르고 나머지는 동일한 것을 확인할 수 있다.

```
183  void Problem2() //The function is solution of task 2
184  {
185      int A[] = { 1,2,3,4,5,6,7,8,9 };
186      int B[15] = { 0 };
187
188      //////Legend////////
189      cout << endl;
190      textcolor(BLACK, YELLOW);
191      cout << "    ";
192      textcolor(WHITE, BLACK);
193      cout << " :Pivot  ";
194      textcolor(LIGHTGREEN, DARKGRAY);
195      cout << "    ";
196      textcolor(WHITE, BLACK);
197      cout << " :Sub Array"<<endl;
198      ////////////////////
199
200      ///////Worst case//////////
201      cout << "--------------------" << endl;
202      Quicksort(A, 0, (sizeof(A) / 4)-1,1);
203      Info_count = 0;
204      cout << "-------Result-------" << endl;
205      printArray(A, NOP, NOP, sizeArray(A));
206      cout << endl;
207      //////////////////////////
```

```
209      //////Best case///////////
210      cout << "--------------------" << endl;
211      FillArray(B,15);
212      CreateBestcase(B,0,15);
213      Quicksort(B, 0, (sizeof(B) / 4)-1,1);
214      Info_count = 0;
215      cout << "-------Result-------" << endl;
216      printArray(B, NOP, NOP, sizeArray(B));
217      cout << endl;
218      //////////////////////////
219
220      ///////Average case////////
221      cout << "--------------------" << endl;
222      InitializeArray(A, 9);
223      CreateRandomcase(A, 9);
224      Random_Quicksort(A, 0, (sizeof(A) / 4) - 1, 1);
225      Info_count = 0;
226      cout << "-------Result-------" << endl;
227      printArray(A, NOP, NOP, sizeArray(A));
228      cout << endl;
229      //////////////////////////
230      return;
231  }
```

다음 함수는 Problem2에 대한 답으로 각각 case를 step by step으로 나타내는 것이다. 이에 대한 정확한 출력은 2번문제에서 자세하게 설명하겠습니다.

```
233 void Problem3() //The function is solution of task 3
234 {
235     int size = 0;
236
237     cout << "Input your size of Array: "; //Declare size of Array
238     cin >> size;
239
240     int* A = new int[size]; //create array
241
242     FillArray(A, size); //Fill Array with items -> Worst case
243     P_flag = 1;
244     textcolor(LIGHTRED, BLACK);
245     printArray(A, NOP, NOP, size); //print Array before sorting
246     P_flag = 0;
247     Quicksort(A, 0, size - 1, NOP); //sorting
248     cout << "--------------------------------" << endl;
249     printArray(A, NOP, NOP, size); //print Array after sorting
250     cout << "--------------------------------" << endl;
251     cout << "Number of Worst comparisons: " << Info_count << endl << endl << endl;
252     Info_count = 0;
253
254     FillArray(A, size); //Fill Array with items
255     CreateBestcase(A, 0, size); //and makes Array of best case to practice quick sort
256     P_flag = 1;
257     textcolor(LIGHTRED, BLACK);
258     printArray(A, NOP, NOP, size);
259     P_flag = 0;
260     Quicksort(A, 0, size - 1, NOP);
261     cout << "--------------------------------" << endl;
262     printArray(A, NOP, NOP, size);
263     cout << "--------------------------------" << endl;
264     cout << "Number of Best comparisons: " << Info_count << endl << endl << endl;
265     Info_count = 0;
```

```
267     InitializeArray(A, size);
268     CreateRandomcase(A, size); //Makes Array of Average case
269     P_flag = 1;
270     textcolor(LIGHTRED, BLACK);
271     printArray(A, NOP, NOP, size);
272     P_flag = 0;
273     Random_Quicksort(A, 0, size-1,NOP);
274     cout << "--------------------------------" << endl;
275     printArray(A, NOP, NOP, size);
276     cout << "--------------------------------" << endl;
277     cout << "Number of Best comparisons: " << Info_count << endl << endl << endl;
278     Info_count = 0;
279     delete[] A;
280
281     return;
282 }
```

위 함수는 Problem3에 대한 답이 제시되는 함수로써 최종적으로 각 case별로 비교횟수를 계산하여 출력할 수 있도록 설계되었습니다. 비교 횟수는 partition에서 pivot이 자리를 찾기 위해 사용되므로 전역변수를 통한 count를 해당 함수에 넣어준 후, sorting이 모두 끝나면 출력되도록 설계되었습니다. 마찬가지로 3번 문항을 다룰 때 좀 더 정확하게 다루겠습니다.

```
284 void Problem4() //--> Equal to Problem 3, just different measure content
285 {
286     chrono::system_clock::time_point StartTime, EndTime; //value of checking excution time
287     chrono::microseconds excution;
288
289     excution = chrono::duration_cast<std::chrono::microseconds>(EndTime - StartTime);
290
291     int size = 0;
292     int repeat = 0;
293
294     cout << "Input your size of Array: ";
295     cin >> size;
296
297     cout << "Input your number of repeating: ";
298     cin >> repeat;
299
300     Info_Worst = new int[size];
301     Info_Best = new int[size];
302     Info_Average = new int[size];
303
304     for (int i = 0; i < repeat; i++)
305     {
306         int* A = new int[size];
307
308         FillArray(A, size);
309         P_flag = 1;
310         textcolor(LIGHTRED, BLACK);
311         printArray(A, NOP, NOP, size);
312         P_flag = 0;
```

```
313         ////////////////////////////////////////////////////////////////////////////////
314         StartTime = chrono::system_clock::now(); //Information of current time
315         Quicksort(A, 0, size - 1, NOP);
316         EndTime = chrono::system_clock::now();
317         excution = chrono::duration_cast<std::chrono::microseconds>(EndTime - StartTime);
318         Info_Worst[Info_count2] = excution.count();
319         ////////////////////////////////////////////////////////////////////////////////
320         cout << "--------------------------------" << endl;
321         printArray(A, NOP, NOP, size);
322         cout << "--------------------------------" << endl;
323         cout << "Excution time: " << excution.count() << endl << endl << endl;
324         Info_count = 0;
325
326         FillArray(A, size);
327         CreateBestcase(A, 0, size);
328         P_flag = 1;
329         textcolor(LIGHTRED, BLACK);
330         printArray(A, NOP, NOP, size);
331         P_flag = 0;
332         ////////////////////////////////////////////////////////////////////////////////
333         StartTime = chrono::system_clock::now(); //Information of current time
334         Quicksort(A, 0, size - 1, NOP);
335         EndTime = chrono::system_clock::now();
336         excution = chrono::duration_cast<std::chrono::microseconds>(EndTime - StartTime);
337         Info_Best[Info_count2] = excution.count();
338         ////////////////////////////////////////////////////////////////////////////////
339         cout << "--------------------------------" << endl;
340         printArray(A, NOP, NOP, size);
341         cout << "--------------------------------" << endl;
342         cout << "Excution time: " << excution.count() << endl << endl << endl;
343         Info_count = 0;
```

```
345         InitializeArray(A, size);
346         CreateRandomcase(A, size);
347         P_flag = 1;
348         textcolor(LIGHTRED, BLACK);
349         printArray(A, NOP, NOP, size);
350         P_flag = 0;
351         ////////////////////////////////////////////////////////////////////////////////
352         StartTime = chrono::system_clock::now(); //Information of current time
353         Random_Quicksort(A, 0, size - 1, NOP);
354         EndTime = chrono::system_clock::now();
355         excution = chrono::duration_cast<std::chrono::microseconds>(EndTime - StartTime);
356         Info_Average[Info_count2] = excution.count();
357         ////////////////////////////////////////////////////////////////////////////////
358         cout << "--------------------------------" << endl;
359         printArray(A, NOP, NOP, size);
360         cout << "--------------------------------" << endl;
361         cout << "Excution time: " << excution.count() << endl << endl << endl;
362         Info_count = 0;
363         Info_count2++;
364
365         delete[] A;
366         Sleep(1000);
367     }
```

해당 함수는 위 3번 함수와 거의 동일하며, 측정하는 요소가 비교횟수가 아닌 실행 시간이며, 마이크로 초 단위로 측정하였습니다. 또한 간편한 그래프 작성을 위해 표본을 한번에 추출할 수 있도록 반복문과 출력해주는 부분이 따로 존재합니다.

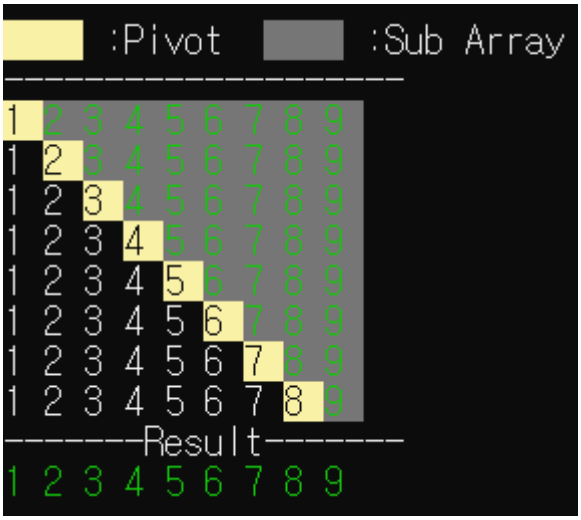
```
418  void CreateBestcase(int Array[], int begin, int end)  //For creating Array of best case
419  {
420      int count = end - begin;
421      if (count < 3)
422          return;
423
424      int middle = begin + (count - 1) / 2;
425
426      CreateBestcase(Array,begin,middle);
427      swap(Array, begin, middle); //Swap first item and middle item
428      CreateBestcase(Array,++middle,end);
429  }
```

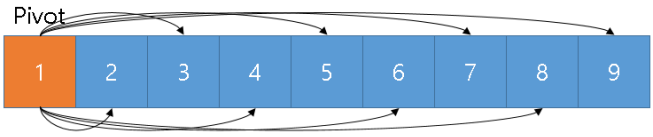
다음 함수는 Best case를 측정하기 위해 Best case에 맞는 array를 생성해주는 함수입니다. 특정 사이트를 이용하여 참고하였으며, 이는 분할이 1/2, 1/2이 되어야 함을 이용하여 오름차순으로 정렬되어있는 배열의 첫 번째 item과 가운데 item을 계속해서 바꿔주는 동작을 실행하게 됩니다. 이렇게 생성된 배열은 sorting할 때 가장 적은 비교와 측정 시간을 갖게 되어야 합니다.

여기까지가 해당 과제에서 쓰인 중요한 함수들이며, 특정 설명되지 않은 배열을 초기화 시켜주는 함수와 swap을 담당하는 함수들은 비교적 간단하고 저번 과제에서도 사용하였기에 설명을 생략했습니다. (후에 값 측정을 위해 출력, 반복 동작이 추가된 부분이 있습니다. 동작 자체는 동일합니다.)

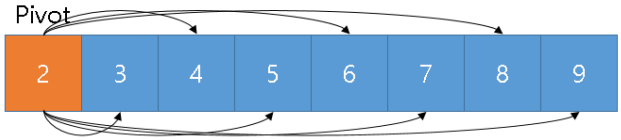
2. When you run your program for the following cases, show the step-by-step results. Explain the results at least four lines.

(1) One example of input sequence for the worst-case time complexity of quicksort





Pivot과 각각의 성분을 비교하여 pivot보다 작은 값, 큰 값으로 Sub array를 나눔

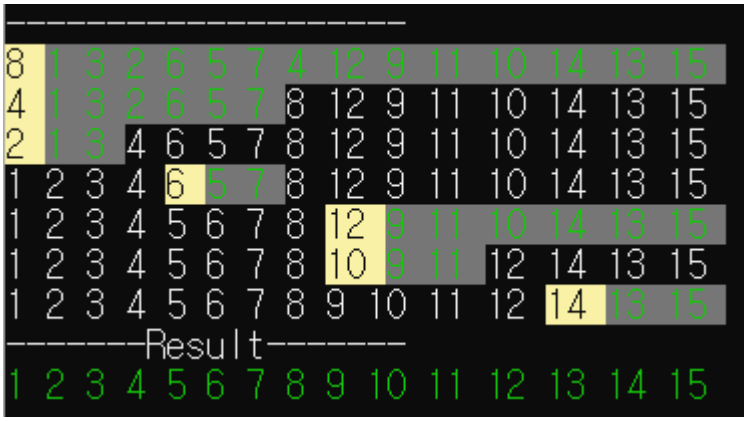


Pivot이 계속해서 뒤 성분들 보다 작으므로 array가 하나만 생성 됨

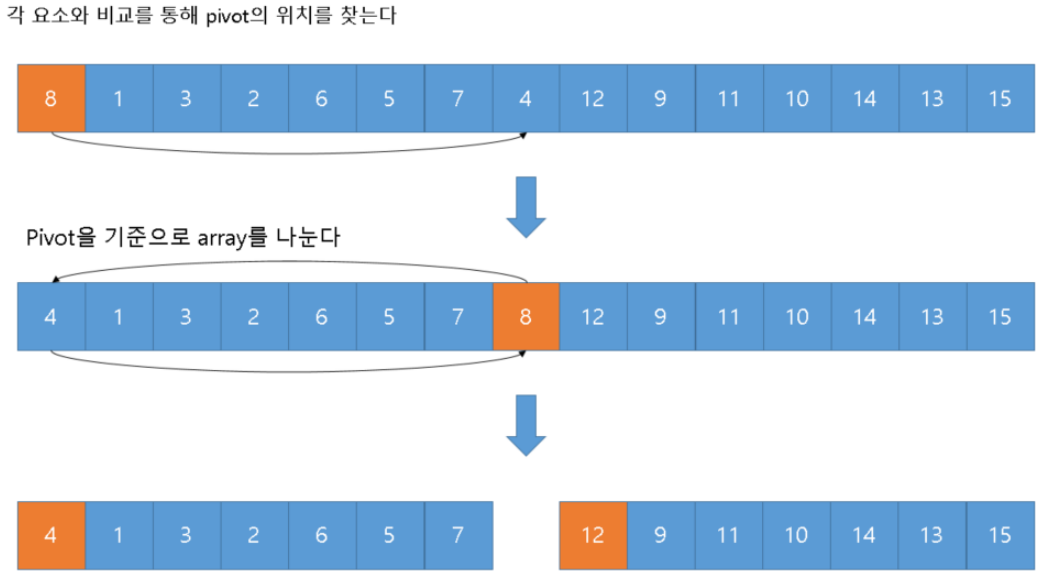
(각 단계에서 위 와 같은 비교가 실행됨)

Quick sort에서 Worst case는 Sub array가 한쪽은 없고 한쪽으로 몰리는 경우다. 위 비교는 Sub array가 한쪽으로 몰린 경우를 나타낸다. 비교를 통하여 pivot이 움직여야 하지만 pivot으로 선택되는 첫 번째 item이 계속해서 가장 작은 성분이므로, 재귀를 통한 Subarray가 계속해서 한쪽으로 몰리는 현상을 확인할 수 있다. 이렇게 되면, 현재 9개의 성분에서 첫 번째 함수 동작에서는 **8번의 비교 다음은 7번의 비교 … 2번의 비교 1번의 비교**로 식으로 나타내면 **(n-1), (n-2), … 2, 1**로 줄어드는 것을 확인할 수 있다. 이는 다시 써서 $\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$ 다음과 같은 식으로 나타낼 수 있으며, $\boldsymbol{\Theta(n^2)}$의 값을 도출할 수 있다.

(2) One example of input sequence for the best-case time complexity of quicksort

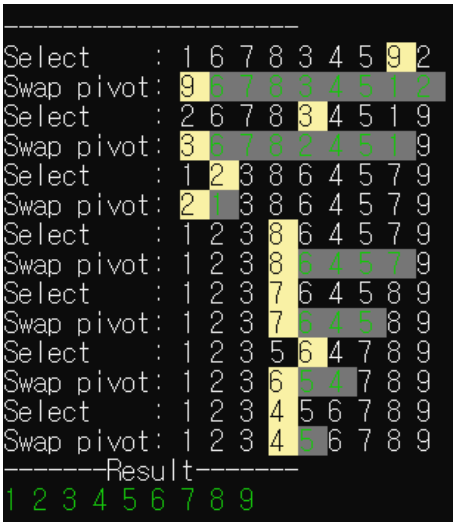


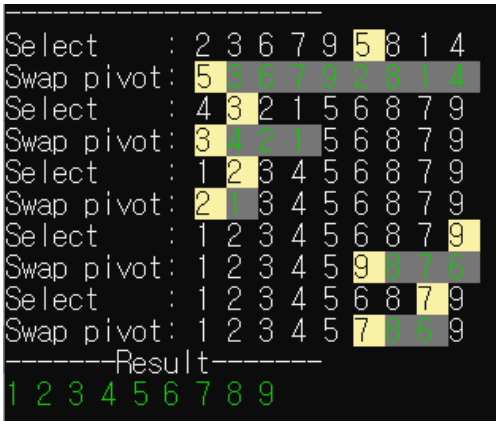
다음은 Best case를 Sorting하는 과정을 step으로 나눈 것이다. 마찬가지로 각 pivot은 array에 해당하는 부분의 모든 요소와 비교과정을 거치며, 왼쪽 Sub array는 pivot보다 작고 반대쪽은 큰 요소들로 배열된다. 첫 번째 줄에서 pivot으로 선택된 8은 다음 줄에서 배열의 가운데로 들어가며, Array를 반으로 나누는 것을 확인할 수 있다. 마찬가지로 두 번째 줄에서 pivot으로 선택된 4또한 3번째줄에서는 해당 Sub array의 가운데로 들어가는 것을 확인할 수 있다. 위 step by step의 동작을 좀 더 자세하게 보면 아래와 같이 나타낼 수 있다.



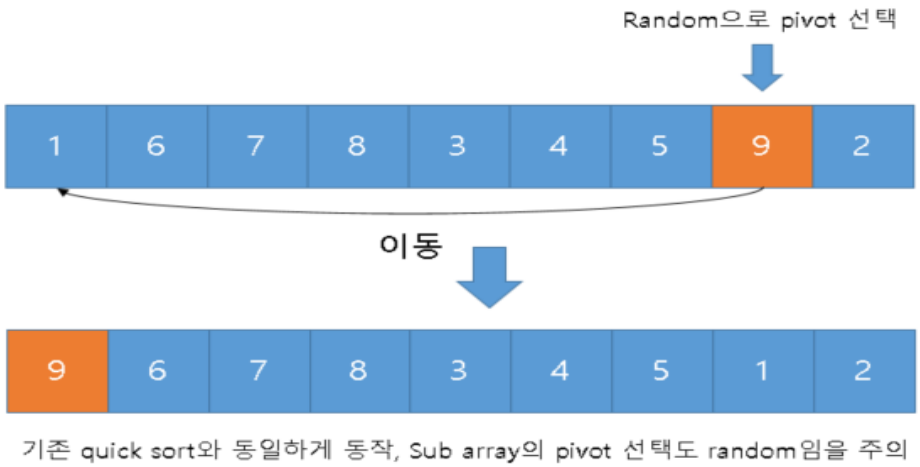
위와 같은 과정을 계속 반복 진행한 것이 위 콘솔에 대한 결과이며, Worst case와 비교 했을 경우 요소개수가 더 많음에도 불구하고 비교횟수가 비슷한 것을 확인 할 수 있으며, 이는 Best case가 확실히 효율적인 것을 알 수 있다.

(3) One random input sequence for the randomized quicksort





다음 두 콘솔 결과는 randomized quicksort를 실행한 size 9개 random input에 대한 결과이다. Randomized quick sort에 경우 worst case를 막기 위해 pivot을 먼저 선택한 후 해당 pivot을 가장 앞 배열로 바꾼 후 기존 정렬을 진행하는 과정으로 pivot을 선택하고 swap후 각 요소와 비교하는 과정으로 나뉜다. 실행 될 때 마다 비교횟수가 일정하지 않지만 요소가 비교되는 횟수자체는 Worst case와 비교하였을 때 확연하게 적은 것을 확인할 수 있다. 확률적으로 Worst case와 같은 횟수가 나오기에는 매우 적은 확률을 갖고 있기에 굉장히 효율적인 알고리즘이다. 아래 그림은 위 콘솔 과정을 자세하게 나타낸 것이다.



Pivot을 기준으로 Sub array가 결정되고 계속 나눠지므로 계속해서 함수를 실행하게 되면 정렬이 되는 것을 확인할 수 있다. 또한 추가적인 작업이 있기에 time complexity가 좋지 않아 보이지만 data의 양이 많아짐에 따라 worst case가 실행되는 경우의 수를 줄이며, Best case complexity에 가까워질 수 있다.

3. When you run your program for the following cases, draw the graphs for the number of comparisons of pivot with elements versus *n*. Explain the graphs at least four lines.

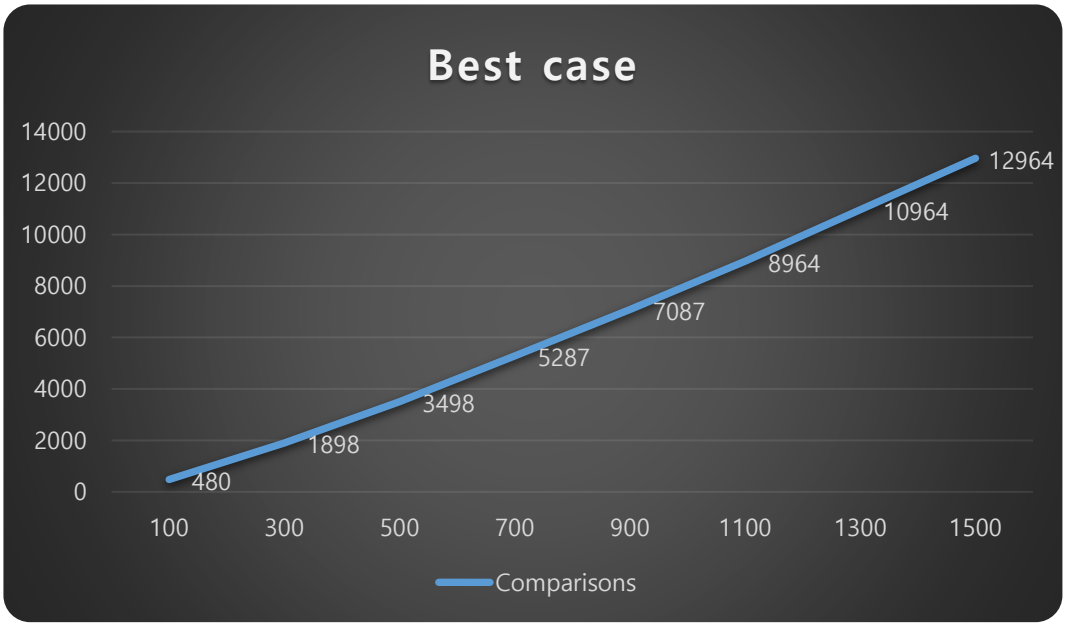
(a) For the worst-case time complexity of quicksort

Input size	100	300	500	700	900	1100	1300	1500
Number of comparisons	4950	44850	124750	244650	404550	604450	844350	1124250

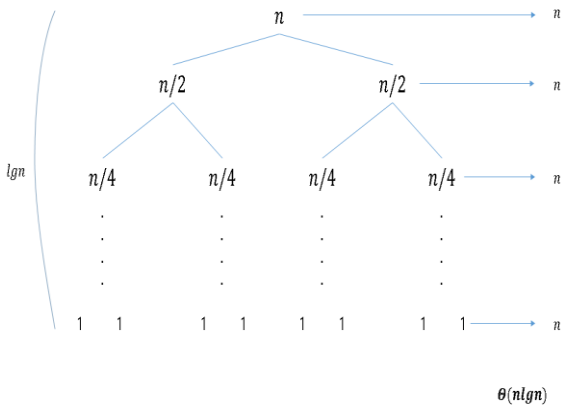
위 결과는 quick sort의 worst case 비교횟수를 그래프로 나타낸 것이다. 비교횟수 규칙 보았을 때 input size n에 대하여 $\frac{n(n-1)}{2}$과 같은 식이 성립함을 알 수 있고(이는 2번 문제에서 도출한 식이다.) 이는 *n*의 값이 커짐에 따라 비교횟수가 기하급수적으로 커짐을 알 수 있다. 식의 값이 *n*의 2차식으로 나오므로 worst case의 Big-theta notation은 n^2 임을 확인할 수 있다 그러므로 $\Theta(n^2)$ → 표기할 수 있다

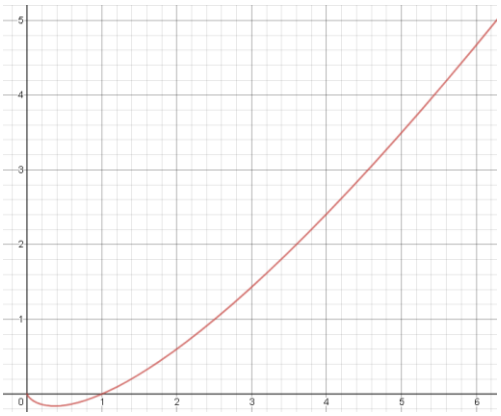
(b) For the best-case time complexity of quicksort

Input size	100	300	500	700	900	1100	1300	1500
Number of comparisons	480	1898	3498	5287	7087	8964	10964	12964



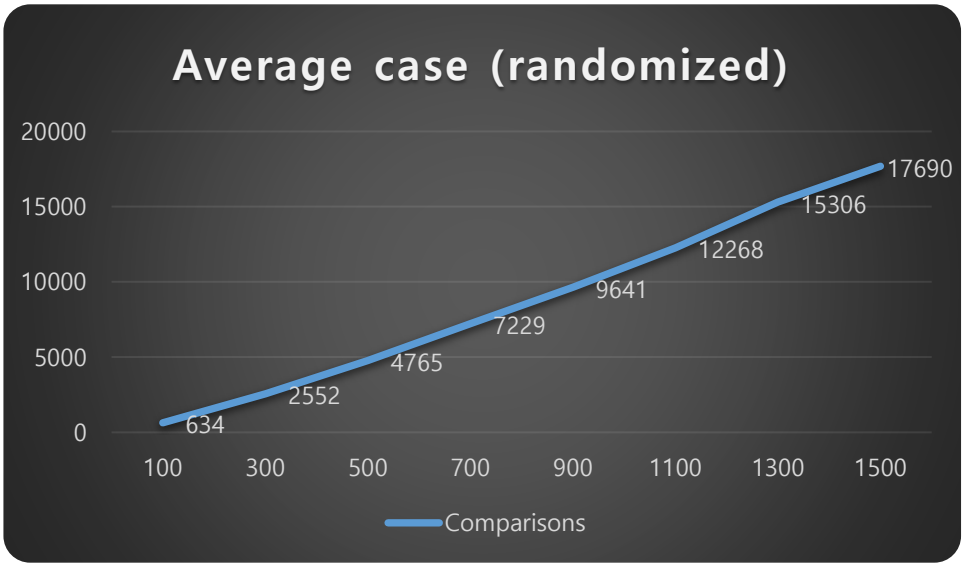
위 그래프는 quick sort의 best case array에 대한 sorting 비교횟수를 그래프로 나타낸 것이다 우선 직관적으로 보기에 complexity의 증가율자체가 많이 낮은 것을 확인할 수 있으며, 이는 수업시간 이론에서 재귀 tree를 사용하여 Big-theta notation에서 *nlogn*의 값을 갖는 것을 확인할 수 있었다. 아래는 복잡도를 재귀 tree로 나타낸 것이며, 그 옆에는 *nlogn*의 실제 그래프를 비교한 것이다.





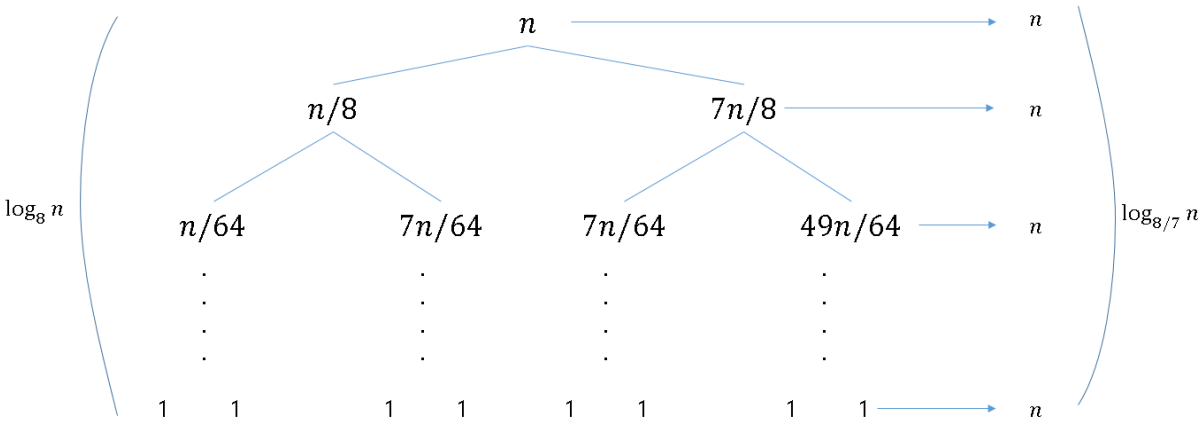
(c) For the average-case complexity of randomized quicksort

Input size	100	300	500	700	900	1100	1300	1500
Number of comparisons(1)	637	2496	4861	7200	9609	12154	18491	18178
2	696	2567	5291	6814	9055	11556	14228	18926
3	618	2351	4303	7297	10136	11908	15655	18446
4	574	2275	4531	7417	11097	12456	14874	18422
5	619	2565	4516	6763	8924	11518	15395	16406
6	603	2626	4474	7171	9304	12377	15355	19344
7	677	2574	5178	7335	8700	12523	15281	16514
8	659	2523	4670	7377	9407	11366	16151	18144
9	583	2445	4813	7017	9021	12869	15758	16955
10	674	2433	4824	7671	9932	11161	15080	18444
11	709	2658	4764	7302	11299	13037	14621	17302
12	693	2797	5088	7569	8893	13199	15525	16652
13	588	2663	4530	6988	9011	12611	14696	16274
14	590	2614	4981	7302	11042	11284	14388	16882
15	589	2690	4658	7206	9179	13997	14090	18458
Average	634	2552	4765	7229	9641	12268	15306	17690



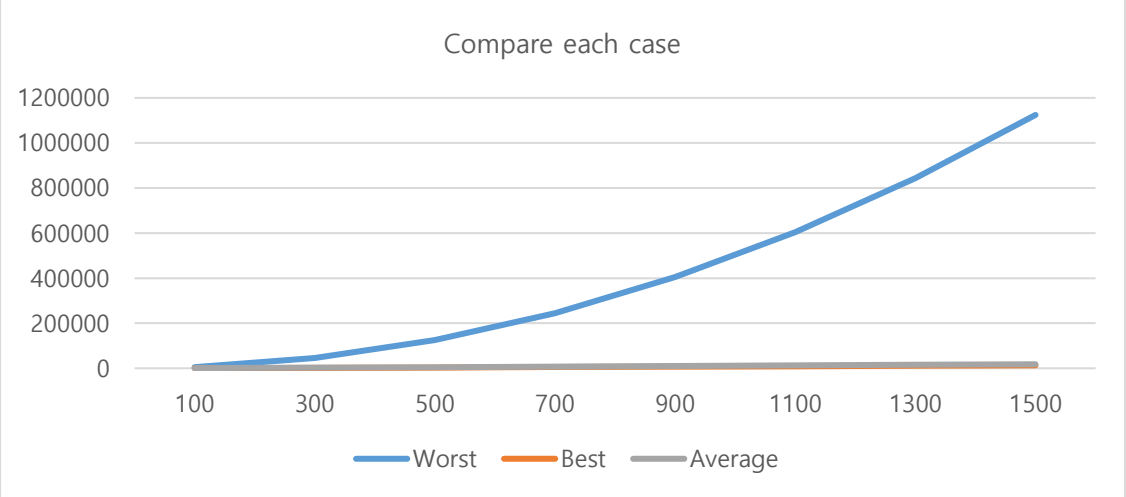
위 결과는 Average case, 즉 array를 무작위로 생성한 후 randomized quick sort를 통한 비교횟수를 나타낸 것이다. Randomized quick sort는 worst case가 실행될 확률을 굉장히 낮춘 개선된 알고리즘으로써, Bubble sort에 경우 worst와 average의 그래프 값이 비슷했다면, quick sort에 경우 Best case와 흡사한 그래프의 모양과 많이 차이 나지 않는 값을 확인할 수 있다. Average case에 경우 Sub array가 한쪽으로 몰리는 경우를 제외한 나머지 경우들이므로 Big-theta notation *nlogn*에 fit한다고 생각할 수 있다.

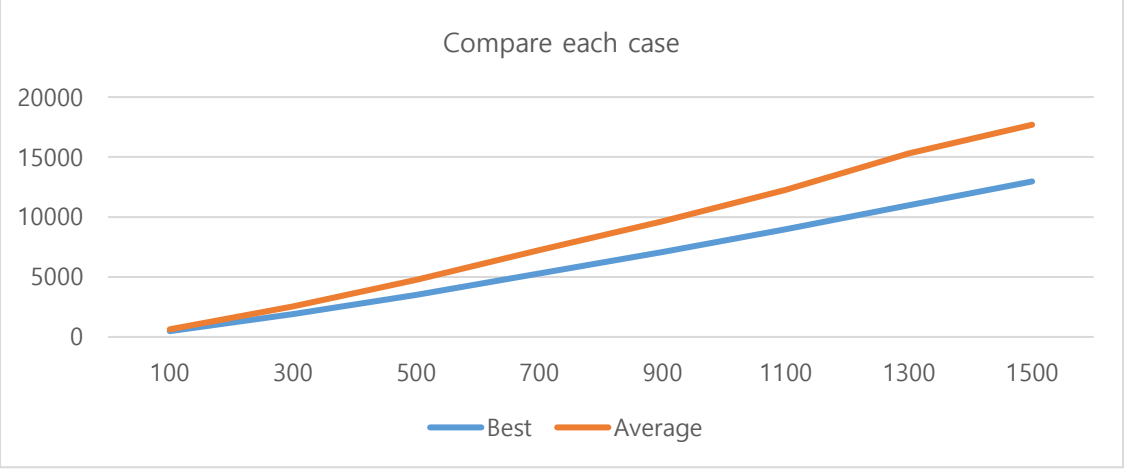
Worst case를 제외한 나머지에 대한 Time complexity



$$log_8 n < lgn < log_{8/7} n \quad \longrightarrow \quad \Theta(nlgn)$$

위와 같이 worst case를 제외한 case중 가장 좋지 않은 case에서도 *nlogn*의 값에 fit하는 것을 확인할 수 있다. 종합적으로 각 그래프를 비교하면 아래와 같이 나타낼 수 있다.





그래프와 같이 worst case가 확연히 큰 값을 갖게 되며 Best와 Average는 거의 비슷하다.

4. When you run your program for the following cases, draw the graphs for the actual running time in your PC versus *n.* Write the basic information about the PC(e.g., CPU, RAM, OS). Explain the graphs at least four lines.

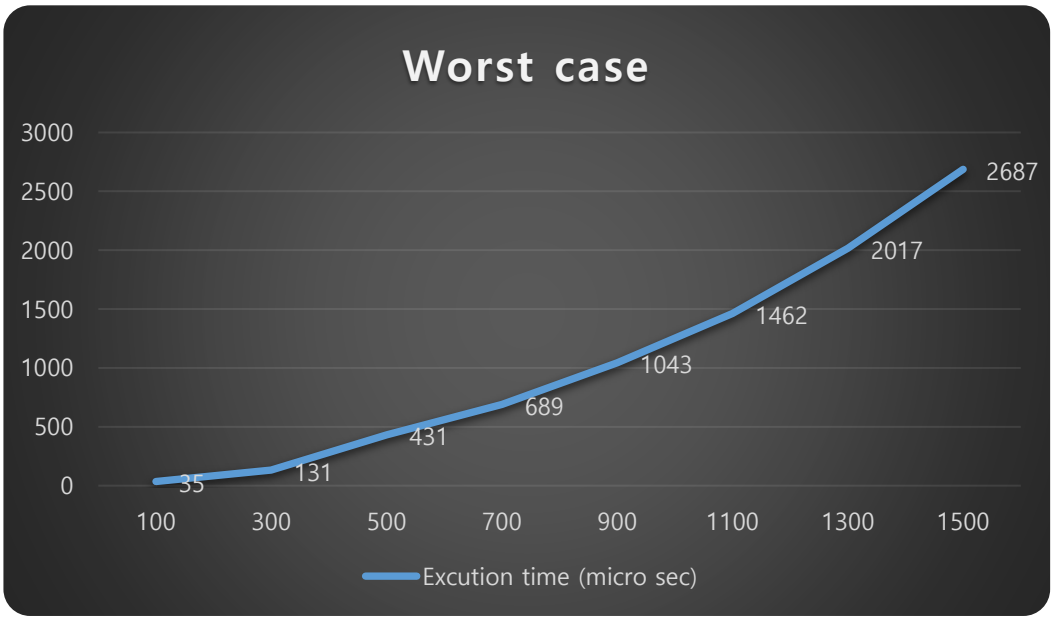
Information about PC:

프로세서	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2501Mhz, 4.
설치된 메모리(RAM):	16.0GB(15.9GB 사용 가능)
OS 이름	Microsoft Windows 10 Home

이전 Bubble sort를 test했던 환경에서 메모리가 변경된 특이점이 있다 전 8GB에서 16GB로 램을 추가했다.

(a) For the worst-case time complexity of quicksort

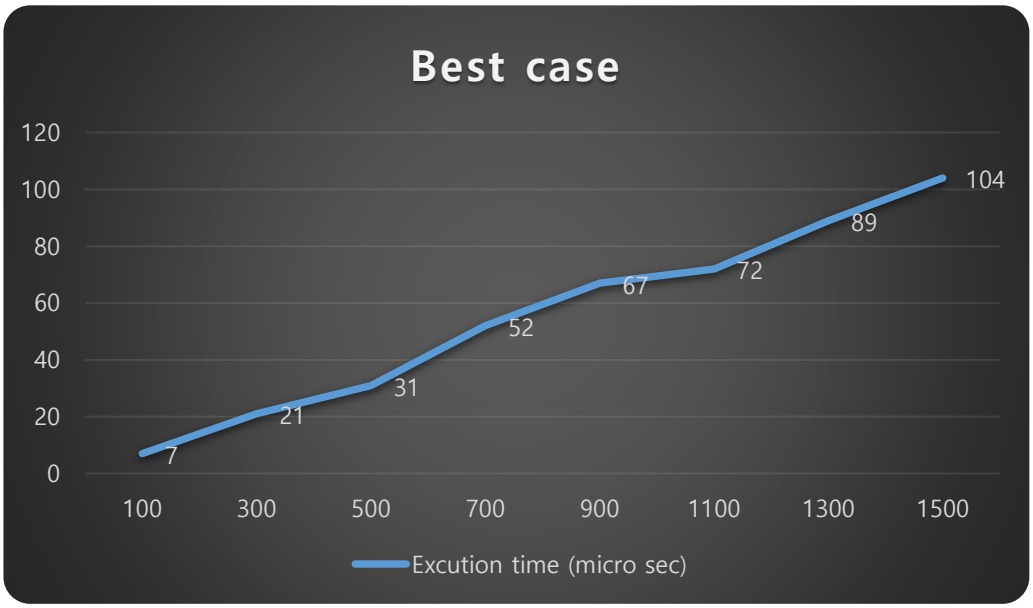
Input size	100	300	500	700	900	1100	1300	1500
Execution time(1)	36	169	678	650	1096	1490	2050	2756
2	25	122	365	658	1040	1449	1991	2650
3	19	122	442	759	1164	1592	1985	2686
4	19	126	488	779	1038	1441	1980	2664
5	36	122	369	774	1140	1440	1986	2661
6	57	123	388	661	1061	1441	2084	2677
7	34	121	368	664	1042	1440	1991	2649
8	25	159	377	668	1065	1448	1998	2646
9	34	122	368	704	1083	1474	2153	2660
10	34	141	495	684	1006	1485	2004	2697
11	32	122	365	679	986	1451	2028	2778
12	38	148	424	657	974	1446	2001	2789
13	78	129	400	673	978	1435	2025	2700
14	38	123	629	671	981	1438	1987	2652
15	23	123	319	664	999	1468	1997	2648
Average	35	131	431	689	1043	1462	2017	2687



위 시간은 Worst case를 측정한 시간이며, 같은 array에서도 측정 환경(ex 음악을 틀어놓는 다는 등..)을 고려하여 몇 번의 반복 진행을 통하여 평균을 구한 값을 측정 값으로 택하였다. 그래프와 결과값을 보면 비교횟수를 나타낸 그래프와 비슷한 모양이며, 값이 기하급수적으로 늘어나는 것을 확인할 수 있다.

(b) For the best-case time complexity of quicksort

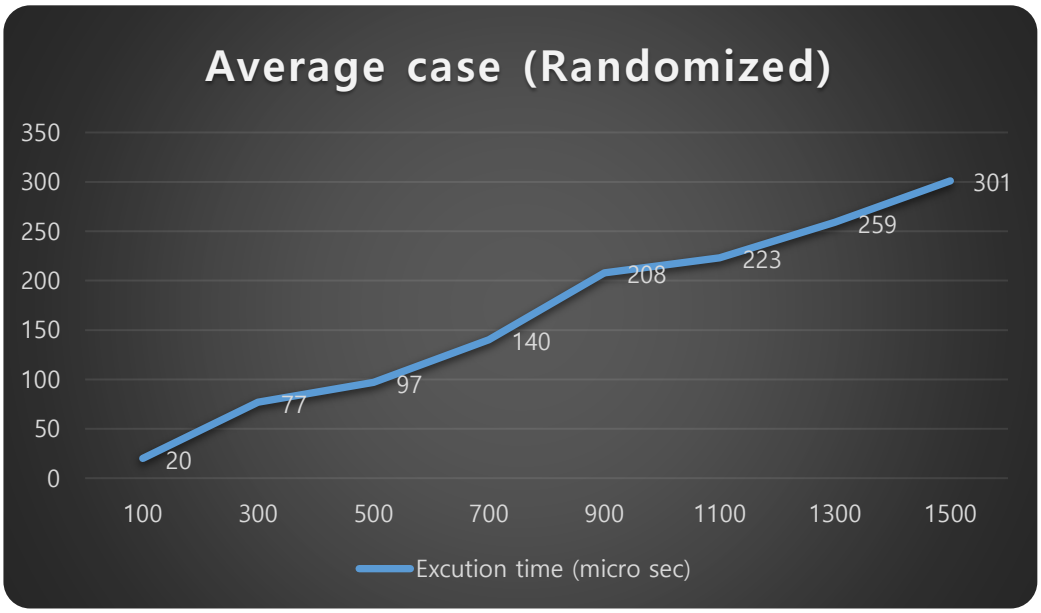
Input size	100	300	500	700	900	1100	1300	1500
Execution time(1)	7	17	27	48	71	68	86	103
2	14	17	27	47	58	69	87	103
3	7	42	27	47	58	68	124	102
4	7	18	38	47	60	85	86	104
5	7	18	29	47	58	68	86	103
6	7	18	34	48	58	69	86	109
7	7	18	34	47	58	77	87	103
8	7	18	27	48	135	69	87	108
9	7	18	39	49	58	69	86	103
10	6	18	27	47	58	68	88	110
11	6	18	28	91	59	100	86	103
12	7	40	40	78	58	68	87	103
13	7	27	35	47	64	69	86	103
14	6	17	28	47	59	68	86	102
15	7	25	27	47	101	69	87	103
Average	7	21	31	52	67	72	89	104



다음은 Best case의 측정시간이다. 시간이 전체적으로 worst에 비해 많이 적게 나오며, 이 또한 똑같지는 않지만 이전 문항에서 다른 비교횟수와 비슷한 모양의 그래프를 따르는 것을 확인할 수 있다.

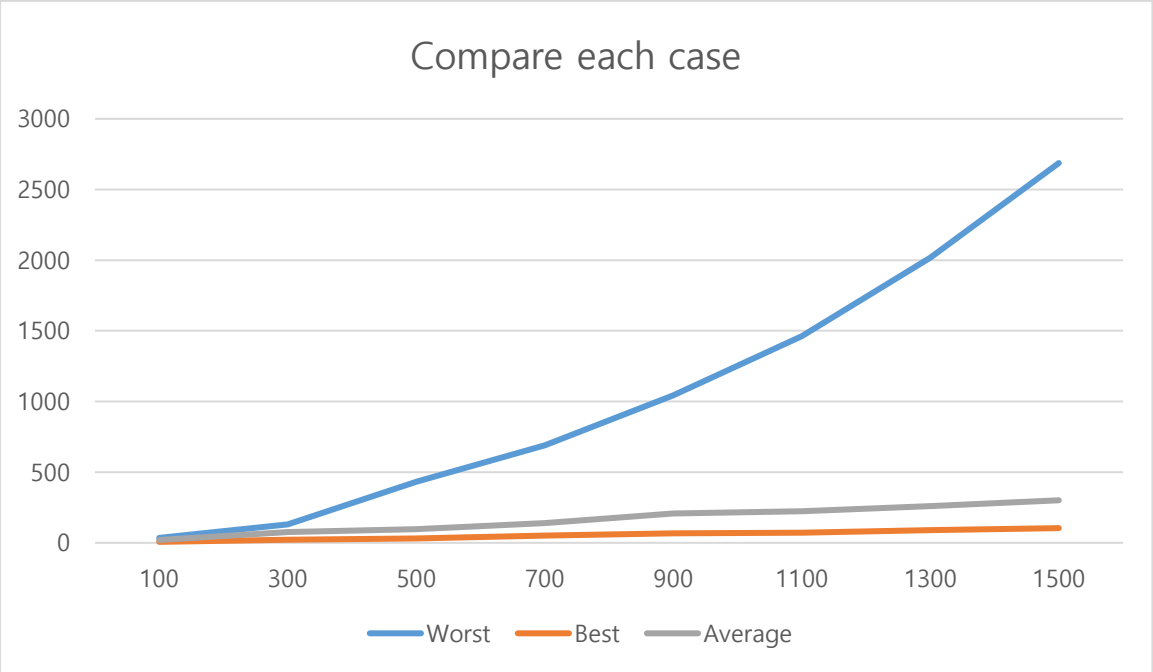
(c) For the average-case time complexity of randomized quicksort

Input size	100	300	500	700	900	1100	1300	1500
Execution time(1)	18	56	95	135	193	255	258	302
2	18	55	96	145	193	227	257	299
3	19	54	95	135	449	2244	260	304
4	19	55	95	140	176	219	256	316
5	19	58	95	136	255	214	259	305
6	19	57	94	147	220	217	259	305
7	18	56	94	145	183	220	255	300
8	18	55	117	134	213	215	279	302
9	18	57	94	134	176	267	261	298
10	31	57	95	145	176	214	256	297
11	18	58	118	174	177	214	261	302
12	19	337	94	136	176	217	258	311
13	17	103	98	136	177	219	261	295
14	18	55	94	134	186	218	255	294
15	37	55	95	135	180	218	262	298
Average	20	77	97	140	208	223	259	301



위 시간은 Average case에 대한 randomized quick sort를 실행한 결과이며, 위 설명과 동일하게 생각할 수 있다.

위에서 비교횟수와 비교하였을 경우 실제 측정시간 또한 비슷한 모양의 그래프와 값의 증가율을 보인다고 할 수 있다. 아래 그래프는 3가지 case에 대한 측정 시간을 비교하기 위한 그래프다.



이전 비교횟수를 비교했던 그래프와 마찬가지로 Worst case같은 경우 다른 두 case에 비해 확연한 차이를 갖고 있음을 알 수 있다. 이는 위에서 설명한 Big-theta notation이 다르다는 점에서 나타나는 차이점이라고 생각할 수 있다.

5. Reference

- ✓ Best case array generator

  https://stackoverflow.com/questions/32556521/java-quicksort-best-case-array-generation

- ✓ Quick sort

  강의 자료 & 온라인 강의

- ✓ Time-measuring idea

  https://jacking.tistory.com/988

- ✓ Working of Randomized Quick sort

  https://slideplayer.com/slide/2810123/