

Problem Set #1 (Algorithms)

Department: 컴퓨터정보공학부

Student ID: 2015722025

Student Name: 정용훈

1. Bubble sort pseudocode

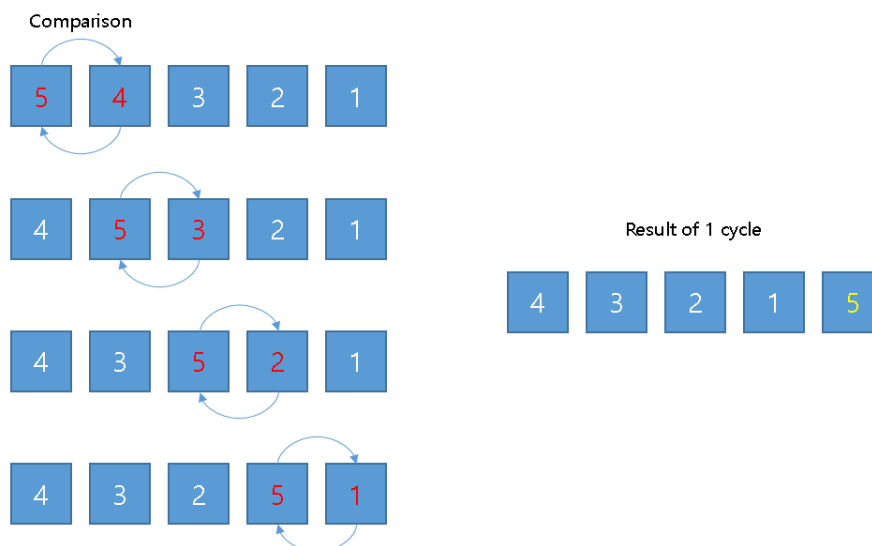
```
for  $i \leftarrow (\text{number of item}) - 1$  to 0 update  $i \leftarrow i - 1$  //Save number of item and repeat until i is zero
{
   $j \leftarrow 0$  &  $\text{flag} \leftarrow 0$  //Save 0 to j and initialize flag to 0
  while  $j < i$  update  $j \leftarrow j + 1$  //Repeat if j is smaller than i
  {
    if  $A[j] > A[j + 1]$  //Practice condition if A[j] is bigger than A[j+1]
    {
      swap  $A[j]$  and  $A[j + 1]$  &  $\text{flag} \leftarrow 1$  //Swap
    }
  }

  if flag is 0 //For checking unnecessary comparison
  {
    end sorting //End of sorting
  }
}
```

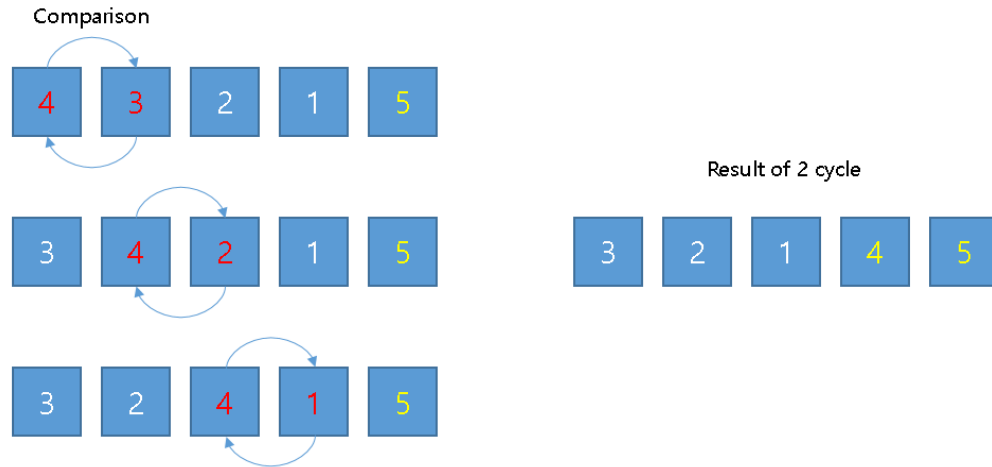
2. Number of comparisons

(1) The worst case

The worst case is when items come in descending order! (When I want array of ascending) Because run all cycle for ordering array. Refer below image.



There are **5** items and they need **4** comparison in **first cycle**. You need refer second image.

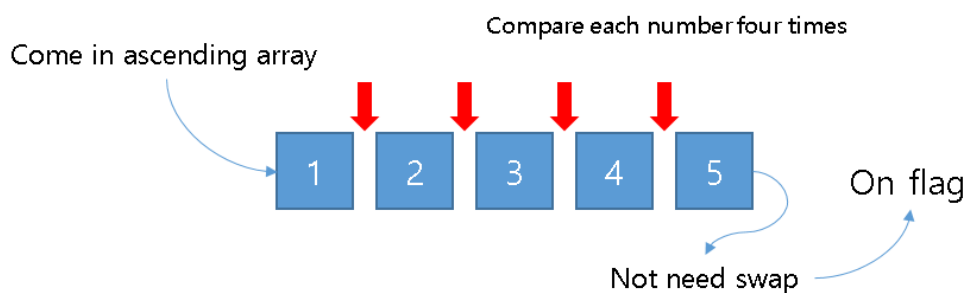


When referring second image, you can see comparison rule of sorting. In worst case you need **(N-1) cycle** and each cycles need **(N-1), (N-2), (N-3).....2, 1** comparison, so generalizing this rule in an equation is as follows.

$$\begin{aligned} \text{Number of comparison} &= (n - 1) + (n - 2) + (n - 3) \dots 3 + 2 + 1 \\ &= n(n - 1) / 2 \end{aligned}$$

(2) The best case

You can weigh the number of comparisons in the best case much more simply. You just need to determine if there is a swap in the array. Refer below image.



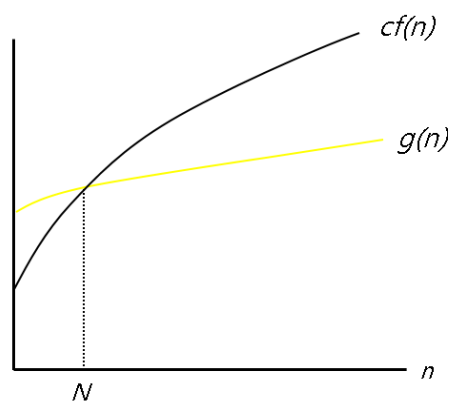
You can use flag if array not need swap, then flag exit function of sorting. In best case (When array come in ascending) just need **N-1** time for comparison.

3. running time in Θ -notation

Before justify running time in big theta (Θ) notation, you need to know big oh (O) and big omega (Ω).

First Big-OH's definition

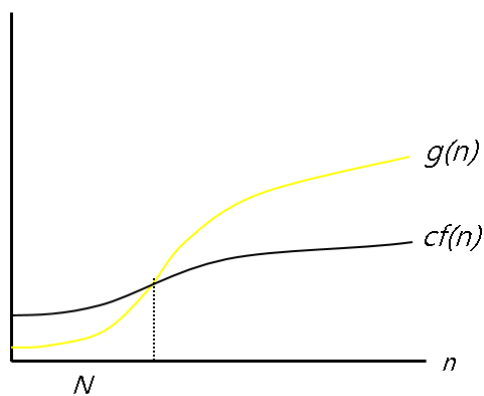
- For a given complexity function $g(n) \in O(f(n))$ satisfies the following
- For all integers n with $n \geq N$, there are real number c ($c > 0$) satisfied $g(n) \leq c \cdot f(n)$ and an integer N , not negative.



(a) $g(n) \in O(f(n))$

Second Big-omega's definition

- For a given complexity function $g(n) \in \Omega(f(n))$ satisfies the following
- For all integers n with $n \geq N$, there are real number c ($c > 0$) satisfied $g(n) \geq c \cdot f(n)$ and an integer N , not negative.

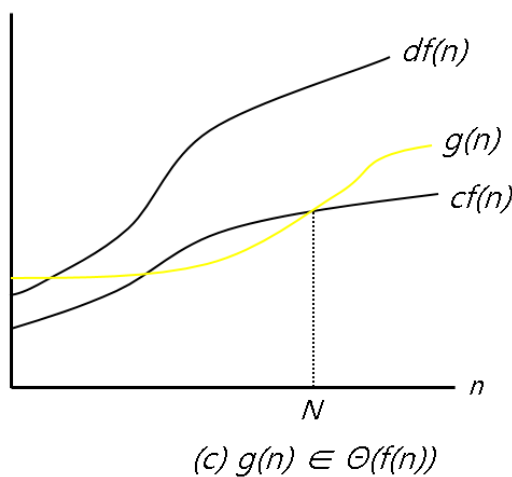


(b) $g(n) \in \Omega(f(n))$

Third Big-theta's definition

Finally you can define Big-theta using the above two definitions.

- For the complexity function $f(n)$, the following is satisfied: $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- So, for all integers n with $n \geq N$, there are real number c ($c > 0$) satisfied $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$ and an integer N , not negative.
- $g(n) \in \Theta(f(n))$: $g(n)$ is the order of $f(n)$



**Ex) $T(n) = (n(n-1))/2$ is $O(n^2)$, and $\Omega(n^2)$.
Therefore, $T(n) = \Theta(n^2)$**

The above definitions of Big-oh and Big-omega show the principles and notation of Big-theta. Then, let's define the running time for each case of the pseudo-code presented in this task.

(1) The worst case

```
for  $i \leftarrow (\text{number of item}) - 1$  to 0 update  $i \leftarrow i - 1$  ---1
{
   $j \leftarrow 0$  &  $\text{flag} \leftarrow 0$  ---2
  while  $j < i$  update  $j \leftarrow j + 1$  ---3
  {
    if  $A[j] > A[j + 1]$  ---4
    {
      swap  $A[j]$  and  $A[j + 1]$  &  $\text{flag} \leftarrow 1$  ---5
    }
  }

  if  $\text{flag is 0}$  ---6
  {
    end sorting ---7
  }
}
```

1: Decrease from N to last 1 to determine if the code proceeds and the repeat ends. (The actual array number is different since the arrangement starts at 0). Thus, the number of repetitions of the code is $n - 2$.

2: Commands inside a repeating statement, which are performed $n - 1$ times because they are executed less than once.

3: The number of times the i value repeats depends on update. " $n + (n-1) \dots 3 + 2$ " equation can be derived, which can be summarized as $\frac{n(n+1)}{2} - 1$. (Be aware that in the actual number of runs, the comparison takes place one more time each.)

4: The fourth code is executed once less each time compared to the number of times the third code is executed, so equations such as " $(n-1) + (n-2) \dots 2 + 1$ " are derived and can be summarized as in $\frac{n(n-1)}{2}$.

5: Since the case is worst case, the conditional statement is executed (swap) every comparison time, so it is executed in the same way as code 4. Therefore, the following expressions are derived $\frac{n(n-1)}{2}$.

6: As with code two, it is located inside the repeating statement one, and the expression of $n - 1$ can be derived.

7: Not used in Worst case.

If all derived expressions are added, $\frac{3}{2}n^2 + \frac{5}{2}n - 2$ will be given, and if the expression is marked as Big-theta, it will be as follows => $\Theta(n^2)$

(2) The average case

To obtain the average execution time, you define the number of times each code can run and divide it by the defined number.

1: The first code says, "1, 2, 3 ... n" run times may occur. Therefore, the first-time formula is as follows: $\sum_{k=1}^n k = \frac{n(n+1)}{2}$ In conclusion, you divide it by the number of items, the final formula is as follows. $\frac{n(n+1)}{2n}$.

2: Since code 2 is executed once less than code 1, the following expression is derived.

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} \rightarrow \frac{n(n-1)}{2(n-1)}$$

3: Code three uses two repetition condition. In order to derive a general formula, the appropriate number of repetitions must be added for each case in code 1. Simply put, for each i-value, you can calculate the average by calculating all the possible times at each i-value.

$$\begin{array}{ccccccc} i & + & i & + & i & + & \dots & + & i & + & i & + & i \\ \swarrow & & \swarrow & & \swarrow & & & & \swarrow & & \swarrow & & \swarrow \\ n+(n-1)\dots 1+2 & & n+(n-1)\dots 1+2 & & n+(n-1)\dots 1+2 & & & & n+(n-1)\dots 1+2 & & n+(n-1)\dots 1+2 & & n+(n-1)\dots 1+2 \end{array}$$

Therefore, the following expressions are derived: $\sum_{k=1}^{n-1} \sum_{i=1}^k (n-i+1)$ When you solve an equation, you can: $\frac{n(n-1)(4n+4)}{12}$ divided by the number of runs $\frac{n(n-1)(4n+4)}{12(n-1)}$

4: Code No. 4 is executed once less than code 3 is executed, and expressions are derived as follows. $(\sum_{k=1}^{n-1} \sum_{i=1}^k (n-i))/(n-1) = \frac{n(n-1)(2n-1)}{6(n-1)}$.

5: For code 5, the code is executed according to the condition after the condition through size comparison has been executed. When averaging is obtained, it is one of the smaller or larger cases, so multiply the general formula by one-half. $\frac{n(n-1)(2n-1)}{12(n-1)}$.

6: Since code 6 is the same number of times as code 2, the formula shall be as follows. $\frac{n(n-1)}{2(n-1)}$.

7: Code 7 belongs to code 6. The average number of execution will be omitted and calculated. (As we use Big-theta notation, it does not have much impact.)

Finally, the largest number of execution times obtained in each code is n^2 , so the Big-theta notation in the Average case is $\Theta(n^2)$

4. My program and comments

The program items have been prepared so that they can operate according to their respective problems (5, 6, and 7). First you can check main function.

```
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 5" << endl; //menu of view
        cout << "2.Problem 6" << endl;
        cout << "3.Problem 7" << endl;
        cout << "4.Exit Program" << endl;
        cout << "Select menu: ";
        cin >> button;

        switch (button)
        {
            case 1:
                Problem5(); //Function of problem 5
                break;
            case 2:
                Problem6(); //Function of problem 6
                break;
            case 3:
                Problem7(); //function of problem 7
                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;
        }
    }
}
```

In the main function, you can select the execution of the desired problem. Next, let's analyze the code that functions for each problem.

(1) Problem 5

```
void Problem5()
{
    int Best[] = { 1,2,3,4,5,6,7 }; //For showing Best case
    int Worst[] = { 7,6,5,4,3,2,1 }; //For showing Worst case
    int* Average = new int[7];
    Create_array(Average,7); //For showing average case(Create random array)

    cout << endl << "(a)Best case" << endl;
    Bubble(Best, sizeof(Best) / 4, 1); //sorting

    cout << endl << "(b)Worst case" << endl;
    Bubble(Worst, sizeof(Worst) / 4, 1); //sorting

    cout << endl << "(c)Average case" << endl;
    Bubble(Average, 7, 1); //sorting

    delete[] Average;
}
```

Question 5 is to print step-by-step what the comparison and swap takes place based on one sample for each case. Detailed execution results are presented by describing item 5. Below is an example of a simple practice.

```
(a)Best case
---Input Array---
1 2 3 4 5 6 7
-----
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
---Output Array---
1 2 3 4 5 6 7
Size of array: 7
Number of cycle: 1
Number of comparison: 6
-----
```

(2) Problem 6

```
void Problem6()
{
    int size, repeat, Case;
    cout << "Input size of array: "; //Get size of array
    cin >> size;
    cout << "Input number of repetitions: "; //Get number of repetitions
    cin >> repeat;
    cout << "(1) Best case" << endl; //Get case of array
    cout << "(2) Worst case" << endl;
    cout << "(3) Average case" << endl;
    cout << "Select your case: ";
    cin >> Case;

    int* A = new int[size];
    Info = new int[size];

    for (int count = 0; count < repeat; count++)
    {
        if (Case == 1) //Get array
            Best_create_array(A, size);
        else if (Case == 2)
            Worst_create_array(A, size);
        else if (Case == 3)
            Create_array(A, size);

        Bubble(A, size, 2); //practice sorting

        for (int init = 0; init < size; init++)
            A[init] = 0;

        Sleep(1000); //sleep function to prevent duplication of seed
    }

    cout << endl;
    cout << "Comparison of all case" << endl; //Output to clean up repeated results
    for (int i = 0; i < Info_count; i++)
    {
        cout << i + 1 << ". ";
        textcolor(YELLOW, BLACK);
        cout << Info[i] << " ";
        textcolor(WHITE, BLACK);
        if ((i + 1) % 5 == 0)
            cout << endl;
    }
    cout << endl;

    delete[] A;
    Info_count = 0;
    delete[] Info;
    return;
}
```

First, the size of the array and the number of repetitions are entered to obtain the sample, and the case of the array is selected. Next, by allowing an option to be printed on a function that outputs information, you can perform a function that outputs a sample that outputs a comparison number to the console. Then, the repeated results are compiled and printed.

The execution of a function can output the following results:

선택 C:\WINDOWS\system32\cmd.exe

```
*****
*      Start Program      *
*      Algorithms        *
*      2015722025        *
*      Jeong yong hoon    *
*****

1.Problem 5
2.Problem 6
3.Problem 7
4.Exit Program
Select menu: 2
Input size of array: 1000
Input number of repetitions: 30
(1) Best case
(2) Worst case
(3) Average case
Select your case: 3
```

```
---Input Array---
689 48 905 178 640 205 41 554 224 707 873 750 238 471 478 241 109 520 182 458 287 887 36 185 693 685 161 67 446 20 105 2
4 102 192 557 738 895 799 767 875 550 187 125 890 878 427 147 345 696 155 571 485 847 489 94 673 866 403 343 472 505 275
53 737 423 854 814 186 674 827 459 156 373 776 937 521 351 610 845 652 532 691 318 784 763 585 416 152 300 645 411 360
999 610 936 188 364 47 608 666 189 633 688 153 639 230 463 172 8 850 419 920 319 984 386 723 171 494 46 831 822 499 361
73 292 867 468 769 88 301 349 731 705 721 961 464 424 183 211 93 570 15 908 970 907 456 267 808 917 327 522 293 561 545
692 730 659 535 372 11 751 958 350 543 177 49 417 787 375 486 527 703 944 163 626 986 805 121 675 55 575 203 780 389 13
82 340 529 85 379 134 72 245 70 439 78 553 18 190 963 27 470 835 877 744 353 829 305 263 556 681 133 921 817 413 724 57
752 128 926 124 39 357 578 628 648 75 843 851 382 381 299 855 76 12 145 97 536 830 798 316 206 754 953 513 354 739 524
17 38 785 496 586 651 655 551 61 884 582 894 170 191 812 6 682 660 789 320 432 232 506 409 838 290 579 919 488 141 945 4
0 5 35 762 376 166 609 540 497 195 257 135 840 646 512 422 294 434 222 728 175 138 607 383 560 502 159 504 940 64 865 8
7 122 547 483 803 266 746 590 962 528 118 844 137 982 593 959 225 380 377 261 457 523 765 939 699 985 896 408 402 704 9
115 207 952 934 823 889 717 596 249 239 220 598 331 199 677 490 98 480 710 690 584 772 858 59 240 322 630 671 297 448
10 784 781 624 149 202 976 462 465 77 569 86 399 790 308 250 394 587 853 778 538 924 862 89 562 777 981 549 809 420 716
405 335 412 474 756 918 874 627 743 44 144 774 964 761 632 365 398 269 426 950 680 227 229 503 993 344 395 886 925 795 4
95 826 793 216 173 129 111 282 436 718 123 113 2 509 753 184 302 87 214 922 712 833 979 54 218 274 204 629 954 546 315 4
16 807 500 237 538 387 949 273 42 589 17 510 22 181 942 142 931 139 594 469 882 428 481 311 672 644 143 378 66 740 278
77 306 28 700 975 9 782 120 271 495 859 869 897 244 678 136 146 160 514 804 165 915 828 309 779 210 197 618 909 328 283
947 21 162 734 272 582 110 581 623 517 281 355 755 371 435 969 687 367 383 453 695 791 508 475 956 745 911 484 984 45 1
8 96 667 83 910 602 385 477 967 174 74 390 894 989 414 759 620 276 879 836 298 286 771 601 735 973 698 555 824 442 599 4
58 801 713 544 861 258 254 507 552 637 221 576 683 647 415 613 511 558 914 208 941 256 37 32 988 649 107 991 613 573 80
539 736 641 611 943 736 131 965 71 555 445 820 533 180 65 846 990 228 501 816 326 223 4 226 656 928 788 291 369 148 19
259 732 948 460 68 606 317 978 3 525 912 79 431 450 312 930 725 701 179 676 722 757 455 418 1000 515 167 438 665 449 3
552 382 33 401 815 10 750 258 567 222 274 40 01 977 40 983 877 700 822 708 700 248 610 938 101 376 102 322 240 222 3
```

```
---Output Array---
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267
268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297
298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327
328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387
388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417
418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507
508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537
538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597
598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627
628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657
658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687
```

```
Size of array: 1000
Number of cycle: 959
Number of comparison: 498680
```

```
Comparison of all case
1.497904 2.498122 3.498720 4.498639 5.499290
6.499472 7.498972 8.498122 9.499485 10.499329
11.498680 12.498510 13.498419 14.499422 15.499247
16.498905 17.497484 18.498972 19.497289 20.498680
21.498069 22.498225 23.498972 24.498597 25.498069
26.498680 27.498759 28.499175 29.499175 30.498680
```

You can easily extract the number of comparisons for the average case by executing the following functions.

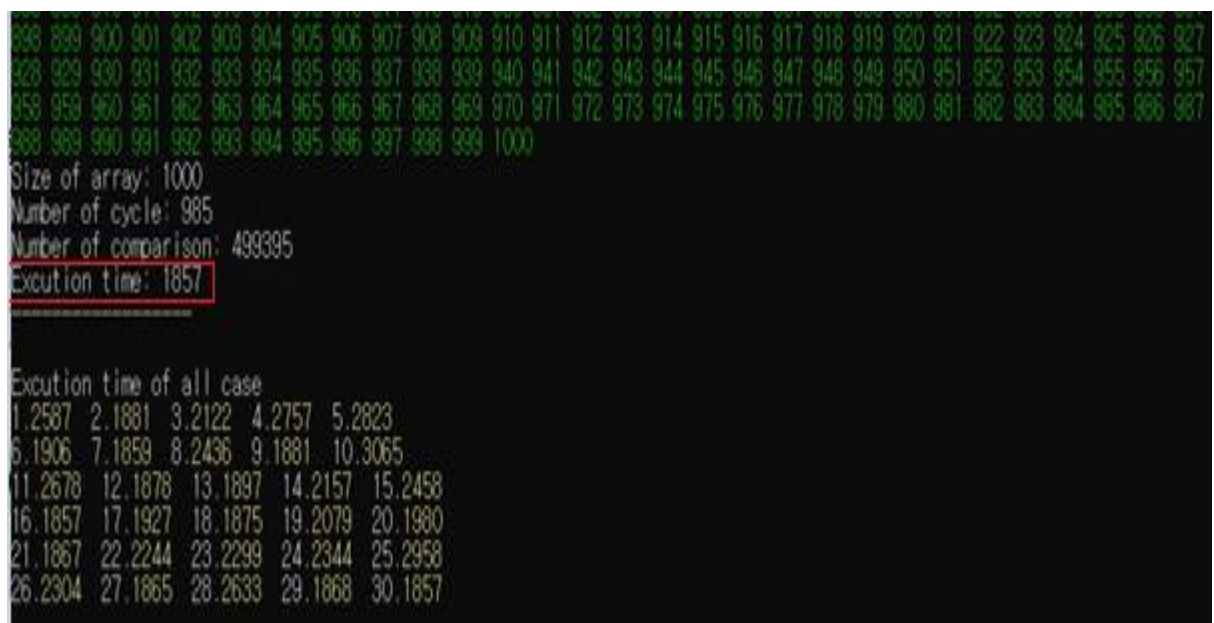
(3) Problem 7

There is a difference in the implementation of question No. 7 and question No. 6. The execution of question No. 7 has an **additional output of the actual execution time**.

You can see that the two functions have different options as factors in proceeding with the Bubble sort. (Option 3 can additionally measure the actual execution time.)

```
Bubble(A, size, 2); //practice sorting
```

```
Bubble(A, size, 3);
```



```
998 999 1000 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927
928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957
958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987
988 989 990 991 992 993 994 995 996 997 998 999 1000
Size of array: 1000
Number of cycle: 985
Number of comparison: 499395
Execution time: 1857

Execution time of all case
1.2587 2.1881 3.2122 4.2757 5.2823
6.1906 7.1859 8.2436 9.1881 10.3065
11.2678 12.1878 13.1897 14.2157 15.2458
16.1857 17.1927 18.1875 19.2079 20.1980
21.1867 22.2244 23.2299 24.2344 25.2958
26.2304 27.1865 28.2633 29.1868 30.1857
```

If you look at the red square, you can see that the actual execution time was added and printed.

(4) Function of Bubble

Next, I will analyze the alignment function used in all the menu.

```
void Bubble(int A[], int size, int option)
{
    int i, j, tmp, flag, comparison = 0, cycle = 0;
    chrono::system_clock::time_point StartTime, EndTime; //value of checking execution time
    chrono::microseconds excution;

    cout << "---Input Array---" << endl;
    textcolor(LIGHTRED, BLACK);
    printArray(A, size, NOP, NOP); //View input array
    textcolor(WHITE, BLACK);
    cout << "-----" << endl;

    if (option == 3)
        StartTime = chrono::system_clock::now(); //Information of current time

    for (i = size - 1; i > 0; i--) //For decreasing time of comparing
    {
        cycle++; //check cycle of bubble sort
        for (j = 0, flag = 0; j < i; j++, comparison++) //For comparing all items
        {
            if (option == 1)
                printArray(A, size, j, cycle); //For checking step by step

            if (A[j] > A[j + 1]) //Check integer
            {
                flag = 1;
                tmp = A[j + 1]; //swap
                A[j + 1] = A[j];
                A[j] = tmp;
            }
        }
        if (flag == 0) //For checking condition of orderig
            break;
    }
}
```

Part of sorting

```
if (option == 3) //For printing real time of execution
{
    EndTime = chrono::system_clock::now();
    excution = chrono::duration_cast<std::chrono::microseconds>(EndTime - StartTime);
}

if (option == 3) //Print execution time
{
    cout << "Execution time: " << excution.count() << endl;
    Info[Info_count] = excution.count();
}

else if (option == 2) //print number of comparison
    Info[Info_count] = comparison;

cout << "---Output Array---" << endl; //For printing information of working
textcolor(LIGHTGREEN, BLACK);
printArray(A, size, NOP, NOP);
textcolor(WHITE, BLACK);
cout << "Size of array: " << size << endl;
cout << "Number of cycle: " << cycle << endl;
cout << "Number of comparison: " << comparison << endl;

if (option == 3)
{
    cout << "Execution time: " << excution.count() << endl;
    Info[Info_count] = excution.count();
}

else if(option ==2)
    Info[Info_count] = comparison;

cout << "-----" << endl;
Info_count++;

return;
}
```

It is a function that actually aligns based on the array received, the information output varies depending on the option, and is implemented to include color in the text output from the console for readability.

5. Result of step by step

(a) Worst case

```
(b)Worst case
---Input Array---
7 6 5 4 3 2 1
-----
7 6 5 4 3 2 1
6 7 5 4 3 2 1
6 5 7 4 3 2 1
6 5 4 7 3 2 1
6 5 4 3 7 2 1
6 5 4 3 2 7 1
6 5 4 3 2 1 7
5 6 4 3 2 1 7
5 4 6 3 2 1 7
5 4 3 6 2 1 7
5 4 3 2 6 1 7
5 4 3 2 1 6 7
4 5 3 2 1 6 7
4 3 5 2 1 6 7
4 3 2 5 1 6 7
4 3 2 1 5 6 7
3 4 2 1 5 6 7
3 2 4 1 5 6 7
3 2 1 4 5 6 7
2 3 1 4 5 6 7
2 1 3 4 5 6 7
---Output Array---
1 2 3 4 5 6 7
Size of array: 7
Number of cycle: 6
Number of comparison: 21
-----
```

You can check a worst case of bubble sorting. Worst case was made in reverse order of ascending order, red is the process of making comparisons, and green means item that alignment has been completed. You can see that the expression $n*(n-1)/2$ is formed when looking at the number of comparisons.

(b) Best case

```
(a)Best case
---Input Array---
1 2 3 4 5 6 7
-----
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
---Output Array---
1 2 3 4 5 6 7
Size of array: 7
Number of cycle: 1
Number of comparison: 6
-----
```

The best case can be arranged in ascending order simply. It can be seen that the comparison is made as shown in the results, but the swap does not occur. Then, it can also be seen that the number of comparisons is formed by the expression $(n-1)$.

(c) Random input sequence

```
(c)Average case
---Input Array---
2 6 4 7 1 5 3
-----
2 6 4 7 1 5 3
2 6 4 7 1 5 3
2 4 6 7 1 5 3
2 4 6 7 1 5 3
2 4 6 1 7 5 3
2 4 6 1 5 7 3
2 4 6 1 5 3 7
2 4 6 1 5 3 7
2 4 1 6 5 3 7
2 4 1 5 6 3 7
2 4 1 5 3 6 7
2 4 1 5 3 6 7
2 1 4 5 3 6 7
2 1 4 5 3 6 7
2 1 4 3 5 6 7
1 2 4 3 5 6 7
1 2 4 3 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
---Output Array---
1 2 3 4 5 6 7
Size of array: 7
Number of cycle: 5
Number of comparison: 20
-----
```

The following is the average case, which shows the step for creating an array with a random. It should be noted that the results can change whenever a function is executed because an array has been created with random.

6. Graphs of based on number of comparisons

(a) Worst case

Time unit : micro

N(items)	100	300	500	700	900	1100	1300	1500
Comparisons	4950	44850	124750	244650	404550	604450	844350	1124250

```

---Output Array---
1 2 3 4 5 6 7 8 9 10 11 12
44 45 46 47 48 49 50 51 52
84 85 86 87 88 89 90 91 92
Size of array: 100
Number of cycle: 99
Number of comparison: 4950
-----

```

```

0 261 262 263 264 265 266 267
284 285 286 287 288 289 290
Size of array: 300
Number of cycle: 299
Number of comparison: 44850
-----

```

```

470 471 472 473 474 475 476 477
493 494 495 496 497 498 499 500
Size of array: 500
Number of cycle: 499
Number of comparison: 124750
-----

```

```

656 657 658 659 660 661 662 663
679 680 681 682 683 684 685 686
Size of array: 700
Number of cycle: 699
Number of comparison: 244650
-----

```

```

865 866 867 868 869 870 871 872
888 889 890 891 892 893 894 895
Size of array: 900
Number of cycle: 899
Number of comparison: 404550
-----

```

```

1078 1079 1080 1081 1082 1083
1096 1097 1098 1099 1100
Size of array: 1100
Number of cycle: 1099
Number of comparison: 604450
-----

```

```

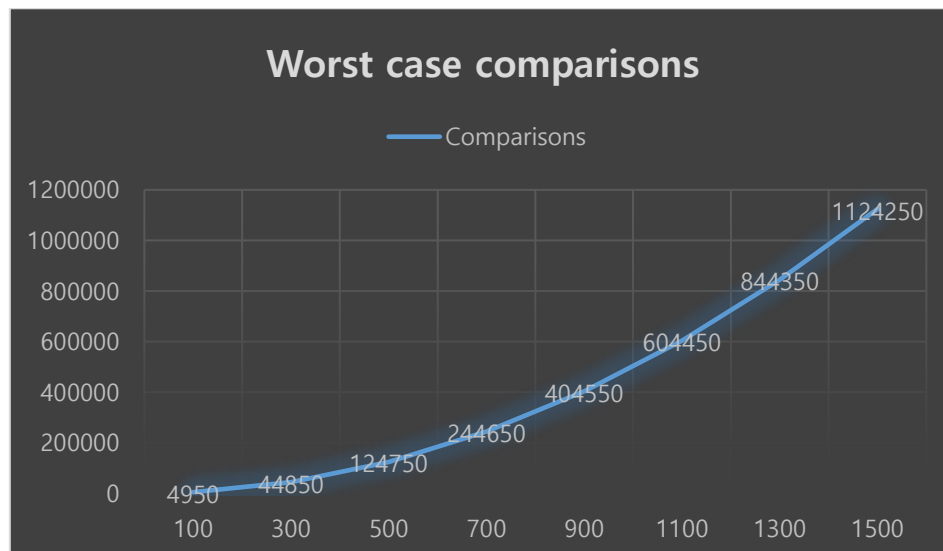
1264 1265 1266 1267 1268 1269
1282 1283 1284 1285 1286 1287 1288
Size of array: 1300
Number of cycle: 1299
Number of comparison: 844350
-----

```

```

1468 1469 1470 1471 1472 1473 1474
1487 1488 1489 1490 1491 1492 1493
Size of array: 1500
Number of cycle: 1499
Number of comparison: 1124250
-----

```



The above table and results are shown in the Worst case. The expression showing the number of comparisons in the Worst case was $n*(n-1)/2$ and as the value of N increases, it can be seen that the value of comparisons increases in the graph form of N^2 .

(b) Best case

N(items)	100	300	500	700	900	1100	1300	1500
Comparisons	99	299	499	699	899	1099	1299	1499

```
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Size of array: 100
Number of cycle: 1
Number of comparison: 99
-----
```

```
261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
Size of array: 300
Number of cycle: 1
Number of comparison: 299
-----
```

```
470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499
Size of array: 500
Number of cycle: 1
Number of comparison: 499
-----
```

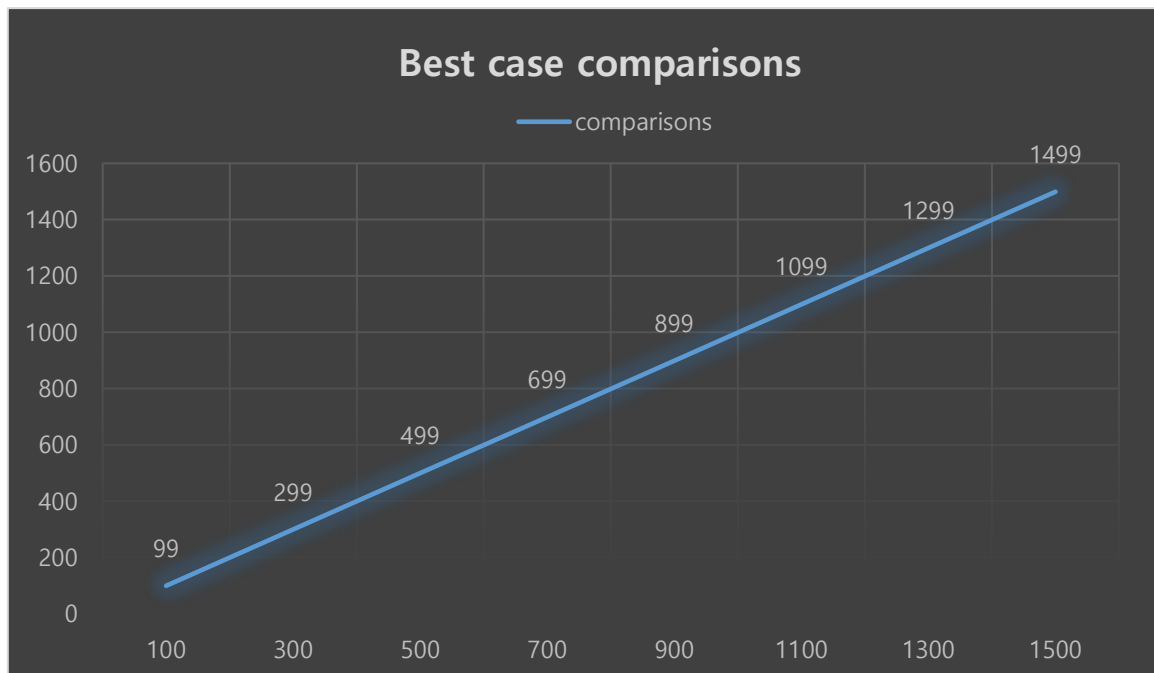
```
656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699
Size of array: 700
Number of cycle: 1
Number of comparison: 699
-----
```

```
865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899
Size of array: 900
Number of cycle: 1
Number of comparison: 899
-----
```

```
1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100
Size of array: 1100
Number of cycle: 1
Number of comparison: 1099
-----
```

```
1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299
Size of array: 1300
Number of cycle: 1
Number of comparison: 1299
-----
```

```
1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499
Size of array: 1500
Number of cycle: 1
Number of comparison: 1499
-----
```



Similarly, the above table and results show the best case. The number of comparisons in the best case is $(N-1)$, which shows that the actual graph also has a straight line graph.

(c) Average case

N(items)	100	300	500	700	900	1100	1300	1500
Comparisons(1)	4779	44759	124560	244089	403604	604125	844245	1119200
2	4929	44840	124254	244154	404360	603589	839097	1123785
3	4947	44550	123319	244055	403689	603820	843972	1123754
4	4940	44679	124672	244185	400085	603124	843489	1122597
5	4895	44772	124009	244530	404115	602910	843755	1123430
6	4929	44772	124714	244299	404199	602370	841424	1122480
7	4905	44784	124372	243660	403065	604099	841190	1123304
8	4895	44385	121095	244584	401229	604314	844197	1123754
9	4940	44849	122670	244244	403847	603322	843885	1122765
10	4944	44472	123847	244397	404054	603547	843755	1119785
11	4872	44714	124285	243747	400980	603670	841722	1123389
12	4914	44772	123970	244089	404085	603670	840072	1123509
13	4779	44814	123574	244419	403884	602434	843684	1122593
14	4950	44795	124009	243425	403172	602434	842810	1121549
15	4884	44697	124399	244514	402954	602497	842139	1124130
Average	4900	44710	123850	244159	403155	603328	842629	1122668

Comparison of all case
1.4779 2.4929 3.4947 4.4940 5.4895
6.4929 7.4905 8.4895 9.4940 10.4944
11.4872 12.4914 13.4779 14.4950 15.4884

Comparison of all case
1.44759 2.44840 3.44550 4.44679 5.44772
6.44772 7.44784 8.44385 9.44849 10.44472
11.44714 12.44772 13.44814 14.44795 15.44697

Comparison of all case
1.124560 2.124254 3.123319 4.124672 5.124009
6.124714 7.124372 8.121095 9.122670 10.123847
11.124285 12.123970 13.123574 14.124009 15.124399

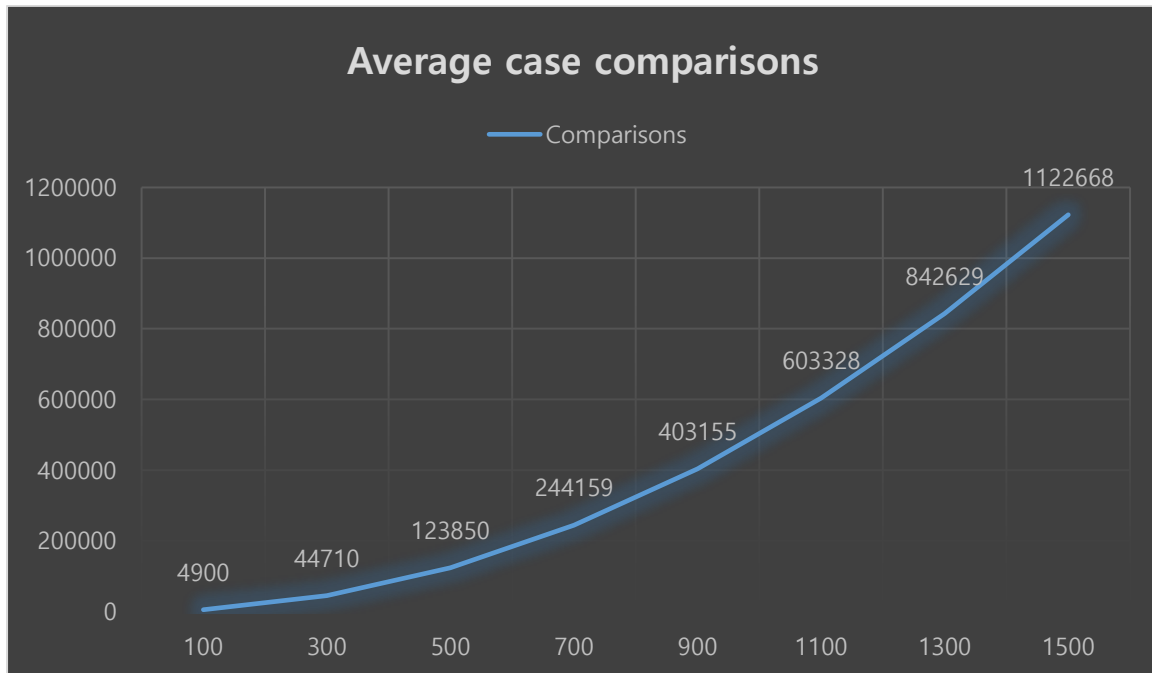
Comparison of all case
1.244089 2.244154 3.244055 4.244185 5.244530
6.244299 7.243660 8.244584 9.244244 10.244397
11.243747 12.244089 13.244419 14.243425 15.244514

Comparison of all case
1.403604 2.404360 3.403689 4.400085 5.404115
6.404199 7.403065 8.401229 9.403847 10.404054
11.400980 12.404085 13.403884 14.403172 15.402954

Comparison of all case
1.604125 2.603589 3.603820 4.603124 5.602910
6.602370 7.604099 8.604314 9.603322 10.603547
11.603670 12.603670 13.602434 14.602434 15.602497

Comparison of all case
1.844245 2.839097 3.843972 4.843489 5.843755
6.841424 7.841190 8.844197 9.843885 10.843755
11.841722 12.840072 13.843684 14.842810 15.842139

Comparison of all case
1.1119200 2.1123785 3.1123754 4.1122597 5.1123430
6.1122480 7.1123304 8.1123754 9.1122765 10.1119785
11.1123389 12.1123509 13.1122539 14.1121549 15.1124130



The above table and the results are for the average case (random case) and it can be found that the graph shows number of comparison similar to the worst case. The reason why the shape of the graph is similar to the Worst case is because the same notation was obtained by the number of execution obtained in paragraph 3 by the Big-theta notation, which results in a similar number of comparisons. The actual number of comparisons itself is almost the same.

7. Graphs of based on actual execution time

Information about PC:

프로세서 Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2501Mhz, 4.

설치된 실제 메모리(RA... 8.00GB

총 실제 메모리 7.89GB

사용 가능한 실제 메모리 3.14GB

총 가상 메모리 15.7GB

사용 가능한 가상 메모리 7.44GB

OS 이름 Microsoft Windows 10 Home

(a) Worst case

N(items)	100	300	500	700	900	1100	1300	1500
Execution time(1)	21	180	505	973	1615	2526	3347	4372
2	39	178	492	971	1602	2386	3377	4540
3	38	180	505	965	1596	2391	3385	4466
4	43	179	495	972	1604	2391	3365	4483
5	21	184	498	970	1601	2423	3494	4473
6	21	178	495	965	1600	2384	3325	4582
7	31	180	497	1006	1602	2380	3335	4481
8	43	178	497	970	1599	2418	3339	4509
9	27	180	495	966	1600	2421	3331	4420
10	38	180	496	972	1613	2391	3319	4450
11	51	179	495	966	1607	2389	3380	4499
12	21	181	496	969	1596	2399	3337	4482
13	34	178	492	969	1599	2430	3407	4438
14	31	203	492	968	1597	2407	3338	4449
15	39	179	495	971	1606	2382	3365	4509
Average	33.2	181.1	496.3	971.5	1602.5	2407.9	3362.9	4476.9

```
Excution time of all case
1.21 2.39 3.38 4.43 5.21
6.21 7.31 8.43 9.27 10.38
11.51 12.21 13.34 14.31 15.39
```

```
Excution time of all case
1.180 2.178 3.180 4.179 5.184
6.178 7.180 8.178 9.180 10.180
11.179 12.181 13.178 14.203 15.179
```

```
Excution time of all case
1.505 2.492 3.505 4.495 5.498
6.495 7.497 8.497 9.495 10.496
11.495 12.496 13.492 14.492 15.495
```

```
Excution time of all case
1.973 2.971 3.965 4.972 5.970
6.965 7.1006 8.970 9.966 10.972
11.966 12.969 13.969 14.968 15.971
```

```

Excution time of all case
1.1615 2.1602 3.1596 4.1604 5.1601
6.1600 7.1602 8.1599 9.1600 10.1613
11.1607 12.1596 13.1599 14.1597 15.1606

```

```

Excution time of all case
1.2526 2.2386 3.2391 4.2391 5.2423
6.2384 7.2380 8.2418 9.2421 10.2391
11.2389 12.2399 13.2430 14.2407 15.2382

```

```

Excution time of all case
1.3347 2.3377 3.3385 4.3365 5.3494
6.3325 7.3335 8.3339 9.3331 10.3319
11.3380 12.3337 13.3407 14.3338 15.3365

```

```

Excution time of all case
1.4372 2.4540 3.4466 4.4483 5.4473
6.4582 7.4481 8.4509 9.4420 10.4450
11.4499 12.4482 13.4438 14.4449 15.4509

```



The end time can vary in condition of PC, even if the same arrangement is executed sorting repeatedly. So the expiry time of Worst case also collected several samples of execution time in the same array, averaged them out, sorted them into tables, and graphs. Note that the unit of extension time follows microseconds. If you look at the graph in Worst case, it will look similar to the one in which the comparison is graphically represented, which is related to the big-theta notation n^2 .

(b) Best case

N(items)	100	300	500	700	900	1100	1300	1500
Execution time(1)	0	1	1	2	3	3	3	4
2	0	1	1	2	2	3	3	4
3	1	1	2	2	2	3	3	4
4	3	1	2	10	2	3	3	4
5	0	1	1	2	2	3	3	5
6	1	1	1	2	2	7	3	4
7	1	1	1	2	2	6	4	4
8	0	1	1	2	2	3	3	4
9	1	1	1	2	2	3	3	4
10	1	1	1	2	2	3	4	4
11	1	1	2	2	2	3	3	4
12	1	1	1	2	2	5	3	4
13	1	1	1	4	2	3	3	6
14	1	1	1	2	3	3	3	4
15	1	1	1	2	2	3	3	4
Average	0.87	1.	1.2	2.67	2.13	3.6	3.13	4.2

Excution time of all case
1.0 2.0 3.1 4.3 5.0
6.1 7.1 8.0 9.1 10.1
11.1 12.1 13.1 14.1 15.1

Excution time of all case
1.1 2.1 3.1 4.1 5.1
6.1 7.1 8.1 9.1 10.1
11.1 12.1 13.1 14.1 15.1

Excution time of all case
1.1 2.1 3.2 4.2 5.1
6.1 7.1 8.1 9.1 10.1
11.2 12.1 13.1 14.1 15.1

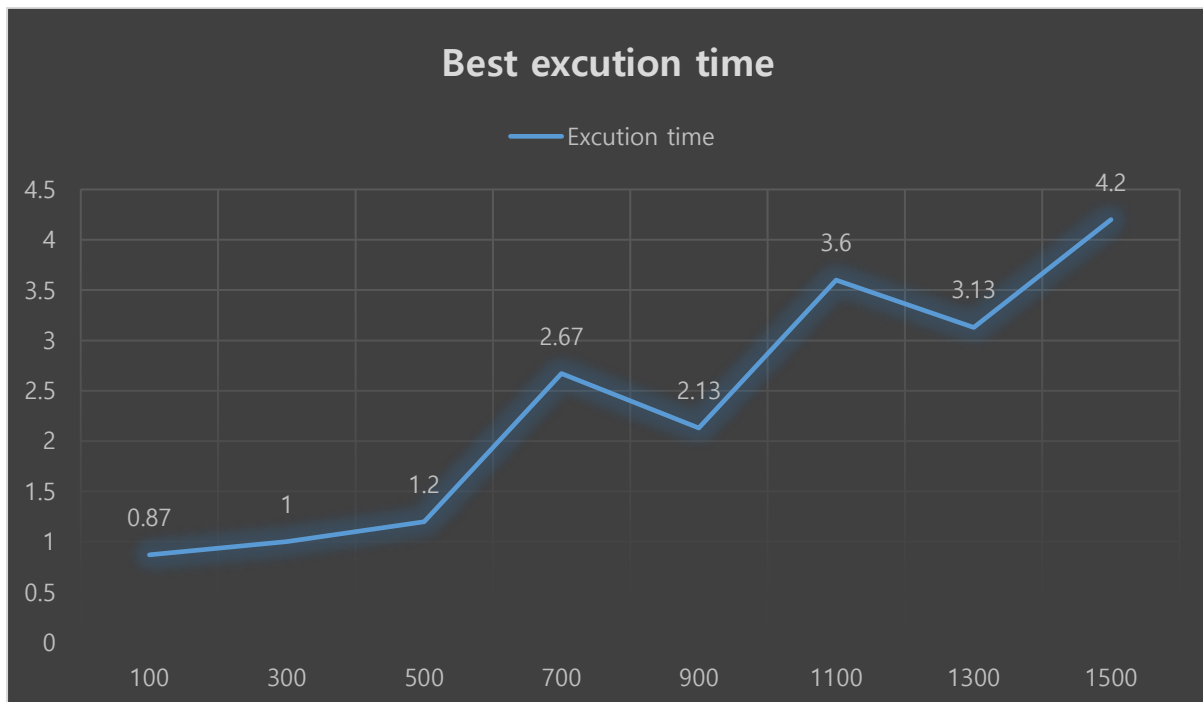
Excution time of all case
1.2 2.2 3.2 4.10 5.2
6.2 7.2 8.2 9.2 10.2
11.2 12.2 13.4 14.2 15.2

Excution time of all case
1.3 2.2 3.2 4.2 5.2
6.2 7.2 8.2 9.2 10.2
11.2 12.2 13.2 14.3 15.2

Excution time of all case
1.3 2.3 3.3 4.3 5.3
6.7 7.6 8.3 9.3 10.3
11.3 12.5 13.3 14.3 15.3

Excution time of all case
1.3 2.3 3.3 4.3 5.3
6.3 7.4 8.3 9.3 10.4
11.3 12.3 13.3 14.3 15.3

Excution time of all case
1.4 2.4 3.4 4.4 5.5
6.4 7.4 8.4 9.4 10.4
11.4 12.4 13.6 14.4 15.4



The following graph shows the best case execution time. Similarly, the average time was calculated and recorded because the aligned array did not always have the same execution time. If you look at the graph, you can see that time appears irregular. I learned that this reason is influenced by the computer environment. For example, if you have a power supply and a laptop battery, you can see that the battery is not only slower, it is much more erratic. However, as a result, you can see a graph that rises when there are more items.

(c) Average case

N(items)	100	300	500	700	900	1100	1300	1500
Execution time(1)	24	259	474	901	1612	2165	4232	4516
2	33	182	470	899	1734	2187	3020	4172
3	43	181	581	1170	1469	2472	3305	4123
4	46	178	463	1392	1458	2428	3057	4010
5	46	185	471	882	1460	2170	3509	4121
6	48	180	466	900	1466	2175	3025	4040
7	73	178	471	901	1470	2198	3088	4068
8	43	323	469	904	1472	2194	3068	4127
9	32	182	467	895	1464	2191	3056	4023
10	54	261	473	900	1458	2200	4316	4204
11	62	176	467	900	1461	2189	3017	4082
12	51	177	469	899	1465	2190	3055	4068
13	39	177	463	903	1529	2199	3021	4205
14	43	175	467	904	1801	2355	3034	4058
15	30	176	473	927	1476	2190	3019	4089
Average	44.5	199.3	476.2	951.8	1519.7	2233.5	3254.8	4127

Excution time of all case
1.24 2.33 3.43 4.46 5.46
6.48 7.73 8.43 9.32 10.54
11.62 12.51 13.39 14.43 15.30

Excution time of all case
1.259 2.182 3.181 4.178 5.185
6.180 7.178 8.323 9.182 10.261
11.176 12.177 13.177 14.175 15.176

Excution time of all case
1.474 2.470 3.581 4.463 5.471
6.466 7.471 8.469 9.467 10.473
11.467 12.469 13.463 14.467 15.473

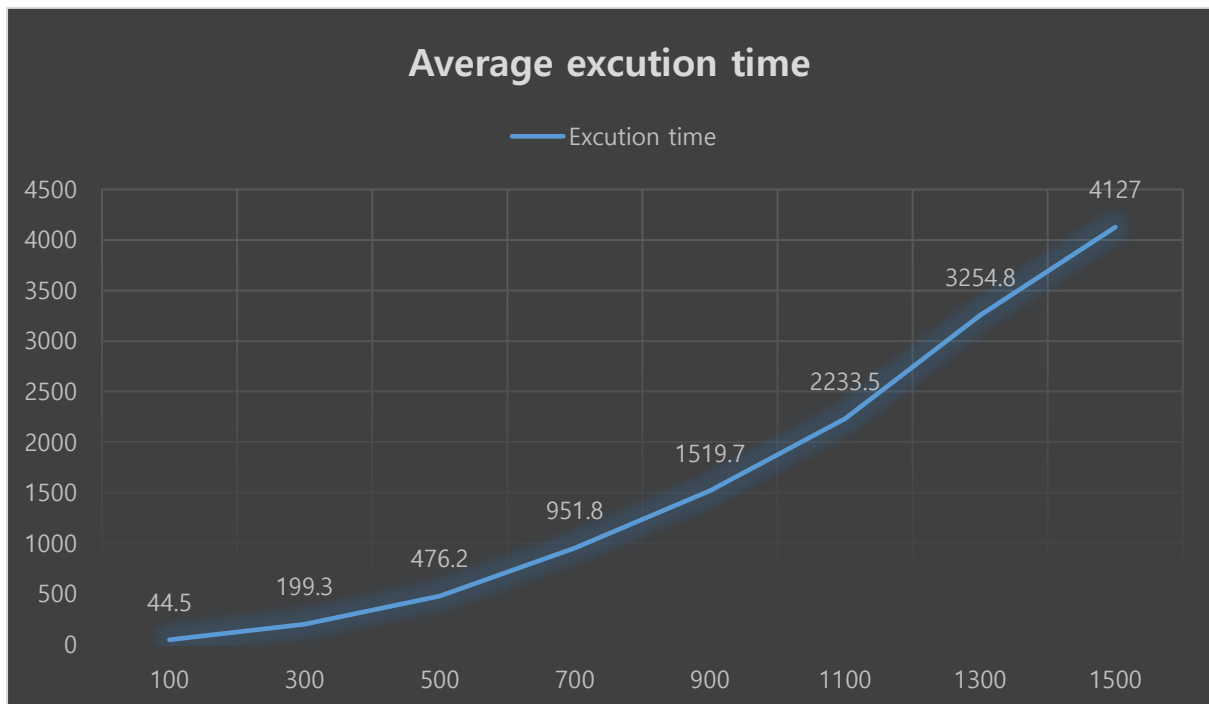
Excution time of all case
1.901 2.899 3.1170 4.1392 5.882
6.900 7.901 8.904 9.895 10.900
11.900 12.899 13.903 14.904 15.927

Excution time of all case
1.1612 2.1734 3.1469 4.1458 5.1460
6.1466 7.1470 8.1472 9.1464 10.1458
11.1461 12.1465 13.1529 14.1801 15.1476

Excution time of all case
1.2165 2.2187 3.2472 4.2428 5.2170
6.2175 7.2198 8.2194 9.2191 10.2200
11.2189 12.2190 13.2199 14.2355 15.2190

Excution time of all case
1.4232 2.3020 3.3305 4.3057 5.3509
6.3025 7.3088 8.3068 9.3056 10.4316
11.3017 12.3055 13.3021 14.3034 15.3019

Excution time of all case
1.4516 2.4172 3.4123 4.4010 5.4121
6.4040 7.4068 8.4127 9.4023 10.4204
11.4082 12.4068 13.4205 14.4058 15.4089



First of all, I did an experiment before writing a report, there was a phenomenon where execution time was more noticeable than Worst case. The next day, when I did the experiment again, I could get less execution time than the Worst case. It has also been learn that this phenomenon is affected by the computer environment (such as battery use) as described above. (Unfortunately, I thought there might be some kind of algorithm or other internal factors, but I couldn't find them). As a result, there is only a slight difference in execution time and the average case also shows a graph of the same shape as the Worst case. This is because the Big-theta notation is the same as n^2 .

8. Reference

- ✓ Expression idea for Average case

<https://www.youtube.com/watch?v=euPIXW7dnI>

- ✓ Time-measuring idea

<https://jacking.tistory.com/988>