

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

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

```
#include <algorithm>
```

```
#include <iostream>
```

```
using namespace std;
```

```
const int n = 1000;
```

```
int insertionCompare = 0;
```

```
int insertionSwap = 0;
```

```
int QuickCompare = 0;
```

```
int QuickSwap = 0;
```

```
int heapCompare = 0;
```

```
int heapSwap = 0;
```

```
long long insertionMoveWeight = 0;
```

```
long long QuickMoveWeight = 0;
```

```
long long heapMoveWeight = 0;
```

```
void insertion(int a[], int N) {
```

```
    int i, j, v;
```

```
    for (i = 0; i < N; i++) {
```

```
        v = a[i];
```

```
        insertionMoveWeight += a[i];
```

```
        j = i;
```

```
        while (j > 0 && a[j - 1] > v) {
```

```
            insertionCompare++;
```

```
            a[j] = a[j - 1];
```

```
            insertionMoveWeight += a[j - 1];
```

```
            insertionSwap++;
```

```
            j--;
```

```
        }
```

```
        a[j] = v;
```

```
        insertionMoveWeight += v;
```

```
        insertionSwap++;
```

```
    }
```

```
}
```

```

void swap(int a[], int i, int j) {
    int t = a[i];
    a[i] = a[j];
    a[j] = t;
}

```

```

int partition(int a[], int l, int r) {
    int i, j, v;
    if (r > l) {
        v = a[l];
        i = l;
        j = r + 1;
        while (1) {
            do {
                ++i;
                QuickCompare++;
            } while (a[i] < v);
            do {
                --j;
                QuickCompare++;
            } while (a[j] > v);
            if (i >= j) break;
            swap(a, i, j);
            QuickSwap += 2;
            QuickMoveWeight += a[i] * 2 + a[j];
        }
        swap(a, j, l);
        QuickSwap += 2;
        QuickMoveWeight += a[j] * 2 + a[l];
    }
    return j;
}

```

```

void quicksort(int a[], int l, int r) {
    int j;
    if (r > l) {
        j = partition(a, l, r);
        quicksort(a, l, j - 1);
        quicksort(a, j + 1, r);
    }
}

```

```
    }  
}
```

```
void makeHeap(int a[], int i, int N) {  
    int j, v;  
    v = a[i];  
    heapMoveWeight += a[i];  
    j = 2 * i;  
    while (j <= N) {  
        if (j < N && a[j] < a[j + 1]) {  
            j++;  
            heapCompare++;  
        }  
        if (v >= a[j]) {  
            break;  
            heapCompare++;  
        }  
        a[j / 2] = a[j];  
        heapSwap++;  
        heapMoveWeight += a[j];  
        j *= 2;  
    }  
    a[j / 2] = v;  
    heapSwap++;  
    heapMoveWeight += v;  
}
```

```
void heapsort(int a[], int n) {  
    int i;  
  
    for (int i = n / 2; i > 0; i--) {  
        makeHeap(a, i - 1, n - 1);  
    }  
    for (i = n - 1; i > 0; i--) {  
        swap(a, 0, i);  
        makeHeap(a, 0, i - 1);  
    }  
}
```

```

int main() {
    int a[n], b[n], c[n];
    for (int i = 0; i < n; i++) {
        a[i] = rand() % 1000;
        b[i] = a[i];
        c[i] = a[i];
        // printf("%5d", a[i]);
    }

    insertion(a, n);
    quicksort(b, 0, n - 1);
    heapsort(c, n);

    cout << "Insertion Compare: " << insertionCompare << endl;
    cout << "Insertion Swap: " << insertionSwap << endl;
    cout << "Quick Compare: " << QuickCompare << endl;
    cout << "Quick Swap: " << QuickSwap << endl;
    cout << "Heap Compare: " << heapCompare << endl;
    cout << "Heap Swap: " << heapSwap << endl;

    cout << "Insertion Move Weight: " << insertionMoveWeight << endl;
    cout << "Quick Move Weight: " << QuickMoveWeight << endl;
    cout << "Heap Move Weight: " << heapMoveWeight << endl;

    // print a[i]
    for (int i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
}

```

코드 출력 결과

```

jw101@DESKTOP-006E1AA MINGW64 ~
$ /usr/bin/env c:\Users\jw101\vscode\extensions\ms-vscode.cpptools-1.12.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe --stdin-Microsoft-MIEngine-In-313gbqvs.5je --stdou
t-Microsoft-MIEngine-Out-pz0gdj1.24r --stderr-Microsoft-MIEngine-Error-ap5ypxgl.jbt --pid-Microsoft-MIEngine-Pid-n0xz3jun.0au --dbgExe=C:\\mingw64\\bin\\gdb.exe --interpreter=mi
Insertion Swap: 254384
Quick Compare: 14146
Quick Swap: 4866
Heap Compare: 5057
Heap Swap: 10582
Insertion Move Weight: 167501852
Quick Move Weight: 3440641
Heap Move Weight: 5381079
2 3 4 4 7 8 8 9 9 9 10 11 11 12 12 16 18 19 19 20 21 22 22 22 23 23 24 25 26 28 29 29 29 31 31 32 32 36 36 36 37 38 38 39 39 40 41 41 41 42 42 42 44 45 45 50 51 53 54 54 54 56 59 5
9 61 61 61 62 65 68 68 71 72 72 73 73 73 74 76 76 76 76 78 81 81 83 83 85 85 86 87 88 88 89 89 91 94 94 98 99 100 102 103 104 106 107 108 109 110 110 111 112 113 113 114 114 115 116 11
6 117 117 118 119 120 124 125 126 128 130 130 132 133 133 140 141 142 142 143 143 143 143 145 146 146 146 147 149 151 153 153 154 154 155 155 156 157 158 160 162 162 165 165 1
89 169 169 170 170 171 171 172 174 174 175 176 177 178 180 182 182 183 185 186 186 187 187 188 189 190 191 191 192 192 193 193 194 194 196 196 196 197 197 200 201 201 201 203 203
203 206 206 210 213 214 214 214 216 216 217 221 222 222 222 223 224 225 225 227 228 230 231 233 234 235 236 241 241 246 248 250 250 254 254 256 257 257 258 259 260 261 262 263 263
264 265 265 265 265 270 271 271 271 272 273 280 280 282 282 282 283 283 286 286 287 287 288 288 289 289 290 291 292 293 293 293 296 297 297 298 298 300 301 303 304 304 304 30
4 307 309 310 310 311 314 314 314 314 314 315 315 315 316 317 317 318 319 319 319 322 323 323 324 324 325 327 329 330 331 334 334 335 335 336 338 338 343 343 344 344 348 349 351 351 351 351 3
54 354 356 356 356 357 358 358 358 360 360 361 362 362 363 364 364 366 369 370 370 371 372 373 373 376 377 377 381 383 384 385 387 390 392 392 393 393 394 399 402 403 405 406 410 411
411 412 412 414 414 414 415 417 417 417 419 422 422 423 423 424 424 424 424 425 426 427 427 429 429 430 431 431 433 434 435 437 438 440 440 440 442 444 447 448 449 451 452 453 455 456
458 458 459 460 461 462 463 464 465 465 467 467 468 468 468 473 473 475 475 477 478 478 478 479 481 482 483 484 484 484 485 485 485 486 486 488 488 489 489 490 490 492 493 494 497 498 49
9 501 504 504 504 505 506 507 509 509 511 511 512 512 513 513 515 516 516 519 520 520 521 521 524 526 527 528 528 530 530 533 535 536 537 538 538 539 539 539 540 541 541 542 542 542 543 5
44 544 546 547 548 548 549 549 549 550 550 550 551 556 556 557 557 557 557 558 559 560 562 562 566 566 566 566 571 574 575 577 577 578 579 581 585 585 585 586 586 588 589 589 590 590
590 592 592 594 594 596 597 597 599 601 601 601 602 602 602 603 605 605 606 607 608 609 609 610 611 612 617 618 618 618 619 620 620 621 623 624 625 625 626 626 626 627 627 627 628
628 628 630 630 630 630 635 635 636 637 638 638 638 640 642 644 645 647 647 649 649 649 650 651 651 652 652 653 654 655 656 656 658 659 659 660 663 663 663 664 665 668 668 669 669 67
1 672 674 674 674 675 676 677 677 679 679 679 680 682 684 686 686 687 688 689 690 691 691 693 694 695 695 696 696 697 699 700 701 702 702 703 704 705 705 705 706 706 706 707 711 712 712 7
713 713 714 717 718 719 719 722 723 724 724 725 725 726 727 727 729 730 735 735 735 735 737 738 741 742 742 746 746 749 749 751 753 754 754 755 757 757 758 758 758 759 759 759 760 760 761
761 763 764 764 764 767 768 770 772 772 772 774 775 776 778 779 779 782 784 784 787 787 788 789 789 790 791 791 797 799 799 799 801 802 803 806 808 809 812 813 814 814 815 816 816 819 819
824 825 825 826 826 828 828 830 830 830 832 832 833 833 834 834 834 836 837 839 841 842 843 844 845 845 848 849 851 851 852 852 854 856 856 856 859 860 862 865 866 867 868 869 870 870 87
870 870 871 871 875 876 876 878 878 882 882 883 886 887 888 888 889 891 893 894 894 895 896 897 897 899 901 901 901 901 902 903 903 903 904 906 910 910 912 913 913 914 914 924 924 9
925 925 927 927 929 930 931 932 933 933 933 935 936 937 938 939 939 939 941 942 942 943 943 944 945 945 945 946 946 946 947 949 949 950 950 952 955 955 955 956 957 959 959 959 960 962 962
963 963 963 964 965 967 967 970 971 972 972 972 973 973 975 975 976 977 978 978 983 986 986 987 990 991 993 994 994 995 996 997 998 999 1000
jw101@DESKTOP-006E1AA MINGW64 ~
$

```

Complexity 분석

비교와 삽입 횟수를 측정했기에, 어렵지 않게 알 수 있다.

<Insertion Sort>

for 문 안의 while 문 속 insertionCompare++; insertionSwap++;을 보면 알 수 있듯, 최악의 경우 $O(n^2)$ 의 시간복잡도.

Time Complexity : $O(n^2)$

Space Complexity : $O(n)$

<Quick Sort>

Partition Func 를 보면 알 수 있듯, 1 번의 partition 실행 시, parameter 로 들어온 l, r 에 대해

약 r-l 회 정도의 비교 연산을 수행하게 됨.

총 input 배열의 길이를 n 이라 하면, 이는 순환 호출 1 회 당 약 $O(n)$ 의 시간복잡도를 갖는다 할 수 있음.

따라서 값이 한쪽으로 몰려있지 않아 pivot 값을 제대로 잡을 수 있는 대부분의 경우 (최선 or 평균적으로) 순환 호출의 깊이가 $\log(n)$ 이라 할 수 있으므로,

최선/평균의 시간복잡도는 $O(n \log n)$.

값이 한쪽으로 몰려있는 등, pivot 값을 잘 잡지 못해 partition 을 한쪽으로 쏠려 진행하게 되는 경우(최악)의 순환 호출 깊이는 약 $n-k$ 회(n 회) 라 할 수 있으므로,

최악의 시간복잡도는 $O(n^2)$

Time Complexity : Best - $O(n \log n)$ / Average - $O(n \log n)$ / Worst - $O(n^2)$

Space Complexity : $O(n)$

<Heap Sort>

Heap 은 Complete Binary Tree 이므로, 자료가 한 쪽에 몰려 Tree 의 깊이가 비정상적으로 깊어질 일이 없다. 따라서 자료 n 개에 대하여 깊이는 약 $\log(n)$ 이라 할 수 있음.

MakeHeap(Heapify)를 모든 자료에 실행하면 Heap 이 완성되는데, 그렇게 하면 정렬이 끝나므로,

Heapify 의 시간복잡도만 측정하면 해결된다.

Heapify 의 시간복잡도는, Tree 의 깊이에 따라 결정되는데, 깊이가 $\log(n)$ 이므로, 시간복잡도도 $O(\log n)$.

따라서 전체 시간복잡도는 $n * O(\log n) \rightarrow O(n \log n)$

Time Complexity : $O(n \log n)$

Space Complexity : $O(n)$