

<b>Algorithms analysis</b>	Section	01
	Student number	22000080
<b>Homework 2 – Sorting algorithms</b>	Name	Kim, Min Chae

*If your explanation is less informative and insufficient, then you may not get any points.  
Also, you should provide discussion, otherwise you will get penalty.*

#### □ Number of statements

Problem	Insertion sort	Heap sort	Quick sort
Prob 1	500499	8317	999
Prob 2	999	9709	999
Prob 3	253403	9081	2411
Prob 4	254571	9033	2385
Prob 5	64147	9471	1602

#### □ Screenshots of your program running

```

1 #include <iostream>
2 #include <fstream>
3
4 using std::string;
5 using std::ifstream;
6
7 void InsertionSort(int num[]);
8 void QuickSort(int num[], int left, int right);
9 void HeapSort(int num[]);
10 void swap(int* a, int* b);
11 void Build_Max_Heap(int num[]);
12 void Max_Heapify(int num[], int i, int length);
13
14 int count = 0; // count num of statements
15
16 int main(){
17     // put nums in array
18     string filename;
19     int num[1000];
20     int i(0);
21
22     std::cout << "input file name : " << std::endl;
23     std::cin >> filename;
24     ifstream input_file(filename);
25
26     45 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782
27     783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820
28     821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 85
29     8 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 8
30     96 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933
31     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
32     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
33     count of statement is 9471
34
35     kimminchae@gimminchaeui-MacBookPro: AlgoAn % ./a.out
36     input file name :
37     prob1.txt
38     sorttype [0]insertion [1]quick [2]heap :
39     0
40     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
41     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
42     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
43     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
44     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
45     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
46     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
47     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
48     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
49     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
50     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
51     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
52     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
53     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
54     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
55     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
56     700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741
57     742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783
58     784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825
59     826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867
60     868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909
61     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
62     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
63     994 995 996 997 998 999 1000
64     count of statement is 500499

```

#### □ Discussion about the results

I created three sort functions along with the main function, and swap, Build\_max\_Heap and Max\_Heapify functions used for the sorting function.

This program assumes that the txt file containing 1000 numbers is located in the same location as the executable.

It first asks the user the name of the file to be read, reads the number of the file, and stores it in an array. It then calls the sorting function by receiving input from the user which of the three algorithms to use. Finally, it outputs a sorted list and the number of statements.

### 1. insertion sort

Its algorithm is simple. It holds number one next, enters back from number zero, and then changes its position if the next is smaller than that. Repeat by increasing the following one by one. Count the next time you change its position. It has  $O(n^2)$  as the worst case and  $O(n)$  as the best case. Therefore, this is  $\Theta(n^2)$ .

### 2. quick sort

It makes use of recursion. It first receives the leftmost and rightmost sides. It makes  $i$  to the left and  $j$  to the right+1 until the left and right positions are normal. Then hold the pivot to the left. If  $i$  is less than or equal to the right and  $num_i$  is less than pivot, continue to increase  $i$ . In addition, if  $j$  is greater than left and  $num_j$  is greater than pivot,  $j$  continues to increase. At this time, if  $i < j$ ,  $num_i$  and  $num_j$  are exchanged. Repeat this until  $i < j$  and switch left and  $j$ . Then recurs the quick sort for the left and right parts. Do the same thing to each partition until all elements are sorted. Its worst case is  $\Theta(n^2)$ . This is when the list is already sorted. The best case is  $\Theta(n \log n)$ . Therefore, its average case is  $O(n \log n)$ .

### 3. heap sort

The heap sort first creates a max heap. At this time,  $O(n)$  takes time. Then from the end, switch it with the first parent and perform Max\_heapify as you go forward. Since the interval for performing Max heap is gradually reduced,  $O(\log n)$  is repeated by  $n-1$ . Therefore, the average time of the heap sort is  $O(n \log n)$ . To illustrate the most important Max\_Heapify function here, after receiving the left and right, make the largest one to the left if it is larger than the  $i$ -th digit on the left. Otherwise, the biggest one is  $i$ . It then calculates largest by comparing the right side in the same way. If the largest is not equal to  $i$ , it recurs by changing the position of  $i$  with the largest.

□ Codes // you should also submit the separate executable C or C++ files, TA will try run your code.

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

using std::string;
using std::ifstream;

void InsertionSort(int num[]);
void QuickSort(int num[], int left, int right);
void HeapSort(int num[]);
void swap(int* a, int* b);
void Build_Max_Heap(int num[]);
void Max_Heapify(int num[], int i, int length);

int count = 0; // count num of statements

int main(){
    // put nums in array
    string filename;
    int num[1000];
    int i(0);

    std::cout << "input file name : " << std::endl;
    std::cin >> filename;
    ifstream input_file(filename);

    if(!input_file.is_open()) {
        std::cout << "file open error" << std::endl;
        return 1;
    }

    while(input_file >> num[i++]);

    input_file.close();

    //sort
```

```

int sorttype;

std::cout << "sorttype [0]insertion [1]quick [2]heap : " << std::endl;
std::cin >> sorttype;

switch (sorttype){
    case 0 : InsertionSort(num);
        break;
    case 1 : QuickSort(num, 0, 1000-1);
        break;
    case 2 : HeapSort(num);
        break;
    default :
        std::cout << "type error" << std::endl;
        break;
}

//print result
for(int j=0; j<1000; j++)
    std::cout << num[j] << " ";
std::cout << std::endl;
std::cout << "count of statement is " << count << std::endl;
return 0;
}

void InsertionSort(int num[]){
    int i, j;
    int next;
    for(i=1; i<1000; i++){
        next = num[i];
        for(j=i-1; j>=0 && next < num[j]; j--){
            num[j+1] = num[j];
            count++;
        }
        num[j+1] = next;
        count++;
    }
}

```

```

    }
}

void QuickSort(int num[], int left, int right){
    int pivot, i, j;
    if(left < right) {
        i = left;
        j = right+1;
        pivot = num[left];
        do{
            do i++; while(i <= right && num[i] < pivot);
            do j--; while(j > left && num[j] > pivot);
            if(i<j){
                swap(&num[i], &num[j]);
                count++;
            }
        } while (i<j);
        swap(&num[left], &num[j]);
        count++;
        QuickSort(num, left, j-1);
        QuickSort(num, j+1, right);
    }
}

void HeapSort(int num[]){
    Build_Max_Heap(num);
    int length = 1000;

    for (int i=1000-1; i>=0; i--){
        swap(&num[0], &num[i]);
        length--;
        Max_Heapify(num, 0, length);
    }
}

```

```

void swap(int* a, int* b){
    count++;
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void Build_Max_Heap(int num[]){
    int i;
    for(int i=499; i>=0; i--){
        Max_Heapify(num, i, 1000);
    }
}

void Max_Heapify(int num[], int i, int length){
    int *tmp;
    int left = 2*i + 1;
    int right = 2*i + 2;
    int largest;

    if((left<length) && (num[left] > num[i]))
        largest = left;
    else
        largest = i;

    if((right < length) && (num[right] > num[largest]))
        largest = right;

    if(largest != i){
        swap(&num[i], &num[largest]);
        Max_Heapify(num, largest, length);
    }
}

```