演算法 hw3

0810546 張理為

Input size = 100

```
Microsoft Visual Studio Debu
Please input the heap_size of the arr:100
6 35 35 37 40 41 82 84 101 106 118 141 145 153 169 190 253 264 281 288 292 299 308 316 322 323 333 334 350 358 370 376 3
82 391 393 421 436 439 446 447 464 467 478 491 500 529 537 538 538 541 547 548 604 623 626 629 644 648 662 664 667 673 7
03 705 711 716 718 723 724 726 729 741 756 757 771 778 805 811 827 827 840 842 859 868 869 890 894 895 902 912 929 931 9
42 942 944 954 961 962 966 995
Execution time: 15 microseconds

D:\nycu_ds_oop\HW3\problem_2\x64\Debug\problem_2.exe (process 27176) exited with code 0.
Press any key to close this window . . .
```

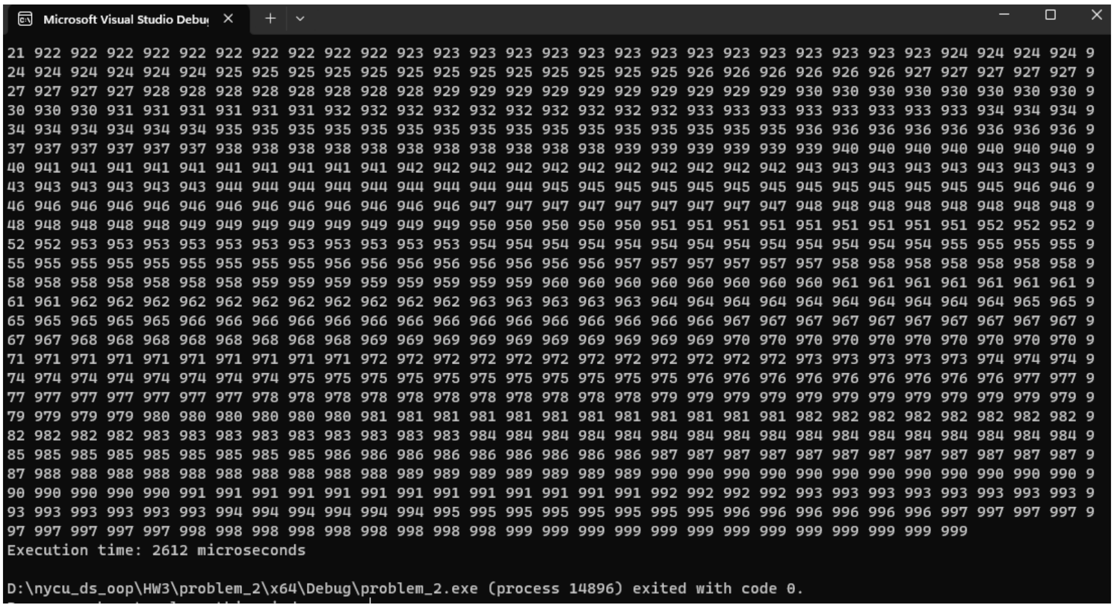
Time duration: 15 microseconds

Input size = 1,000

```
Microsoft Visual Studio Debu
9 230 232 233 234 235 240 240 245 247 249 249 253 253 255 256 256 257 258 259 260 261 262 262 263 264 264 264 264 264 26
9 270 270 270 271 272 279 279 281 281 281 282 282 285 285 286 286 287 287 288 288 289 290 291 292 292 292 292 295 296 29
6 297 297 299 300 302 303 303 303 303 303 306 308 309 309 310 313 313 313 313 313 314 314 314 315 316 316 317 318 318 31
8 321 322 322 323 323 324 326 328 329 330 333 333 334 334 335 337 337 342 342 343 347 348 350 350 350 350 353 353 355 35
5 355 355 356 357 357 357 358 359 359 360 361 361 362 363 363 365 368 369 369 370 371 372 374 375 376 380 382 383 384 38
6 389 391 391 392 392 393 398 401 402 404 405 409 410 410 411 411 413 413 413 414 416 416 416 418 421 421 422 422 423 42
3 423 423 423 424 425 426 426 428 428 429 430 430 432 433 434 436 437 439 439 439 441 443 446 447 448 450 451 452 454 45
5 457 457 458 459 460 461 462 463 464 464 466 466 467 467 467 472 472 474 474 476 477 477 477 478 480 481 482 483 483 48
3 484 484 484 485 485 487 487 488 488 489 489 491 492 493 496 497 498 500 503 503 503 504 505 506 508 508 510 510 511 51
1 512 512 514 515 515 518 519 519 520 520 523 525 526 527 527 529 529 532 534 535 536 537 537 538 538 538 539 540 540 54
1 541 541 542 543 543 545 546 547 547 548 548 548 549 549 549 549 550 555 555 556 556 556 556 557 558 559 561 561 565 56
5 565 565 570 573 574 576 576 577 578 580 584 584 584 585 585 587 588 588 589 589 589 591 591 593 593 595 596 596 598 60
0 600 600 601 601 601 602 604 604 605 606 607 608 608 609 610 611 616 617 617 617 618 619 619 620 622 623 624 624 625 62
5 625 625 626 626 626 626 627 627 627 629 629 629 629 634 634 634 635 636 637 637 637 639 641 643 644 646 646 648 648 64
8 649 650 650 651 651 652 653 654 655 655 657 658 658 659 662 662 662 663 664 667 667 668 668 670 671 673 673 673 674 67
5 676 676 678 678 678 679 681 683 685 685 686 687 688 689 690 690 692 693 694 694 695 695 696 698 699 700 701 701 702 70
3 704 704 704 705 705 705 706 710 711 711 712 712 713 716 717 718 718 721 722 723 723 724 724 725 726 726 728 729 734 73
4 734 734 736 737 740 741 741 745 745 748 748 750 752 753 753 754 756 756 757 757 757 758 758 758 759 759 760 760 762 76
3 763 763 766 767 769 771 771 773 774 775 777 778 778 781 783 783 786 786 787 788 788 789 790 790 796 798 798 798 800 80
1 802 805 807 808 811 812 813 813 814 815 815 815 818 818 823 824 824 825 825 827 827 829 829 829 831 831 832 832 833 83
3 833 835 836 838 840 841 842 843 844 844 847 848 850 850 851 851 853 855 855 855 858 859 861 864 865 866 867 868 869 86
9 869 869 869 870 870 874 875 875 877 878 881 881 882 885 886 887 888 888 890 892 893 893 894 895 896 896 898 900 900 90
0 900 901 902 902 902 903 905 909 909 911 912 912 913 913 913 923 923 924 924 924 926 928 929 930 931 932 932 932 934 93
5 936 937 938 938 938 940 941 941 942 942 943 944 944 944 945 945 945 946 948 948 949 949 951 954 954 954 955 956 958 95
8 958 959 961 961 962 962 962 962 963 964 966 966 969 970 971 971 971 972 972 974 974 975 976 977 977 982 985 985 986 98
9 990 992 993 993 994 995 996 997 998 999
Execution time: 203 microseconds
```

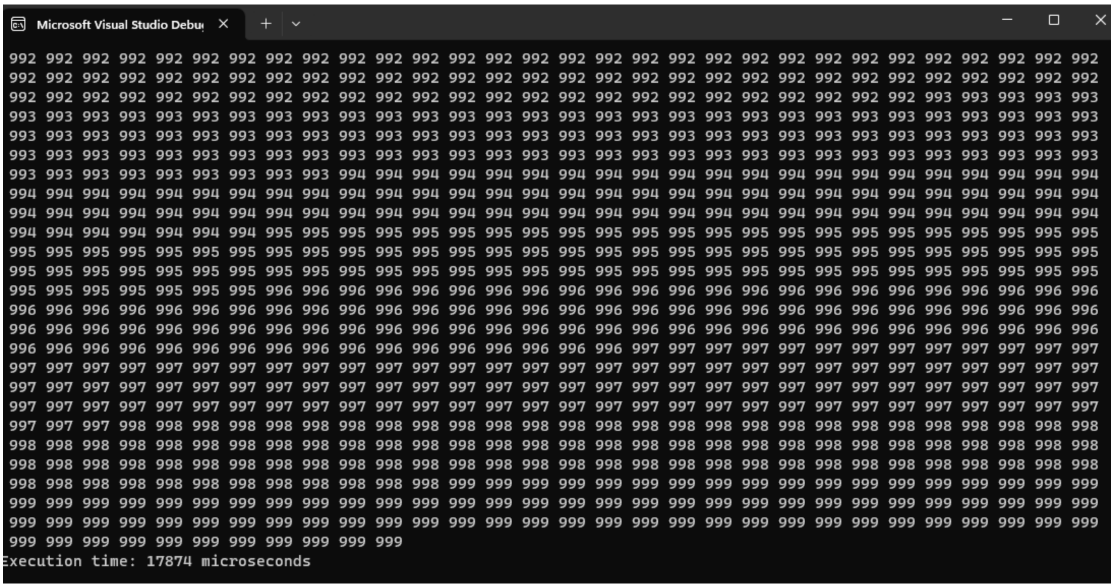
Time duration: 203 microseconds

Input size: 10,000



Time duration: 2612 microseconds

Input size: 100,000



Time duration: 17874 microseconds

將執行時間與 insertion sort 與 merge sort 做比較，可以發現在較少 elements 需排序的時候(n=100)，insertion sort 依然擁有最短的排序時間，然而將 scale 拉大後(n>100)，heap sort 就擁有最短的執行時間。

採用時機:

若需要排序的數目較少時，應該選用 insertion sort 當作排序的方法，若需排序的數目增加，不應當再採取 insertion sort，而是使用 merge sort 或 heap sort，其中 heap sort 的排序時間又會優於 merge sort。

Worst case:

```
int main() {

    int arr_size;
    cout << "Please input the heap_size of the arr:";
    cin >> arr_size;
    int* input = new int[arr_size];
    for (int i = 0; i < arr_size; i++)
    {
        int num = rand() % 1000;
        input[i] = num;
    }


    auto start_time = chrono::high_resolution_clock::now();
    HeapSort(input, arr_size);
    auto end_time = chrono::high_resolution_clock::now();
   // Printarr(input, arr_heap_size);
    auto duration = chrono::duration_cast<chrono::microseconds>(end_time - start_time);
    cout << "Execution time: " << duration.count() << " microseconds" << endl;

    delete[] input;

    return 0;
}
```

分析 HeapSort 的時間複雜度

```
void HeapSort(int* arr, int heap_size) {

    BuildMaxHeap(arr, heap_size);

    for (int i = heap_size - 1; i >= 1; i--) {
        swap(arr[0], arr[i]);
        MaxHeapify(arr, 0, i - 1);
    }
}
```

時間複雜度 = O(BuildMaxHeap) + n-1*O(MaxHeapify) + n-1*O(swap)

BuildMaxHeap:

```
void BuildMaxHeap(int* arr, int heap_size) {

    for (int i = heap_size / 2 - 1; i >= 0; i--) {
        MaxHeapify(arr, i, heap_size - 1);
    }
}
```

第一眼會覺得 O(BuildMaxHeap) = (n/2 -1) * O(MaxHeapify)，然而並非每一個節點都須做 MaxHeapify，最底層的節點（由 n/2 給出）根本不會向下移動。 倒數第二層 (n/4) 的節點將向下移動 1 次，因為下面只剩下一層可以向下移動。倒數第三層的節點將向下移動 2 次，依此類推。

可以得到:

(n/2 * 0) + (n/4 * 1) + (n/8 * 2) + (n/16 * 3) + ...h = n/2 =O(n)

Swap:

```
void swap(int& t1, int& t2) {
    int temp = t1;
    t1 = t2;
    t2 = temp;
}
```

資料交換而已，所以 O(swap)=O(1)

MaxHeapify:

```
void MaxHeapify(int* arr, int root, int heap_heap_size) {

    int left = 2 * root + 1;
    int right = 2 * root + 2;
    int largest;

    if (left <= heap_heap_size && arr[left] > arr[root])
        largest = left;
    else
        largest = root;

    if (right <= heap_heap_size && arr[right] > arr[largest])
        largest = right;

    if (largest != root) {
        swap(arr[largest], arr[root]);
        MaxHeapify(arr, largest, heap_heap_size);
    }
}
```

可以看出跟父節點比較完後，若不滿足條件則會繼續比較，故 worst case 會比比較到整棵樹高，也就是 logn，所以 O(MaxHeapify) = logn

時間複雜度 = O(BuildMaxHeap) + n-1*O(MaxHeapify) + n-1*O(swap)

= O(n) + n-1*O(logn)+n-1

= O(nlogn)

Best case:

當要排序的列表中的所有元素都相同時，則每個節點的運算時間皆為 O(1)，因為不需要將任何節點去做移動或是將最大節點往上移動，因為每個節點的值皆相同，故時間複雜度為 n(n 個節點) x O(1) = O(n)