**알고리즘 설계와 분석 HW2**

컴퓨터공학과 20171666 이예은

**실험 시 사용한 컴퓨터**

OS: Window 10 Home

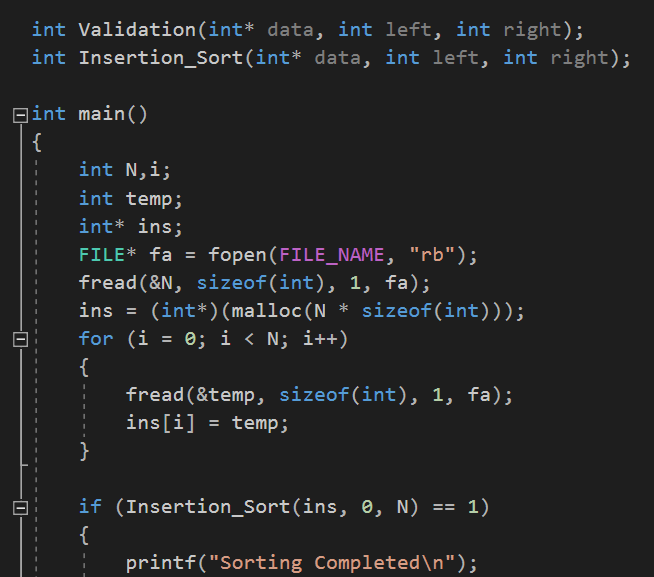
CPU: Intel® Core™ i3-7100U CPU @ 2.40GHz 2.40 GHz

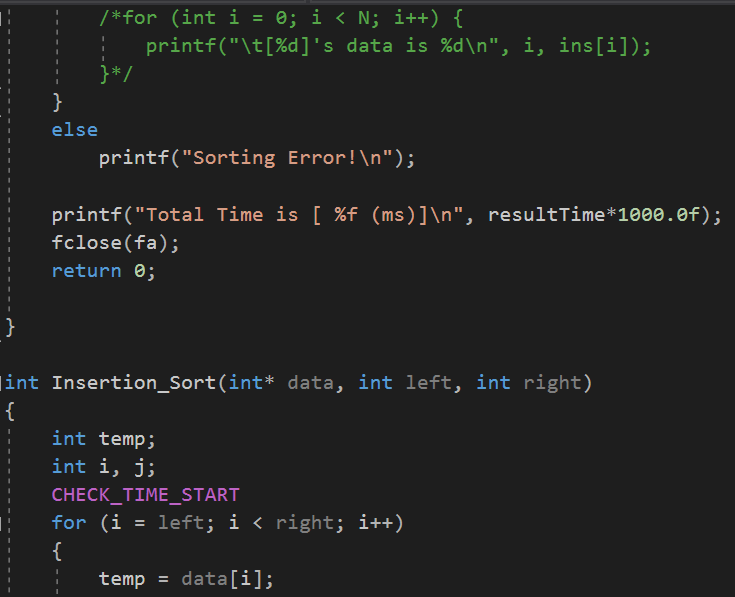
RAM: 4.00GB

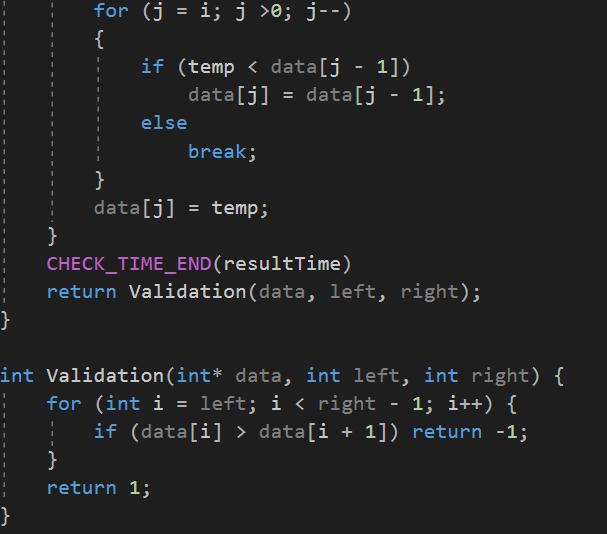
Compiler: Visual Studio 2017 Release mode

**구현 방법.**

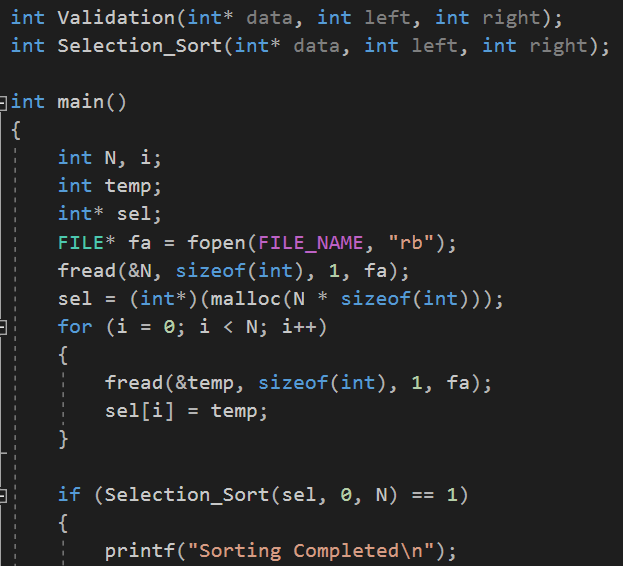
1. Insertion Sort

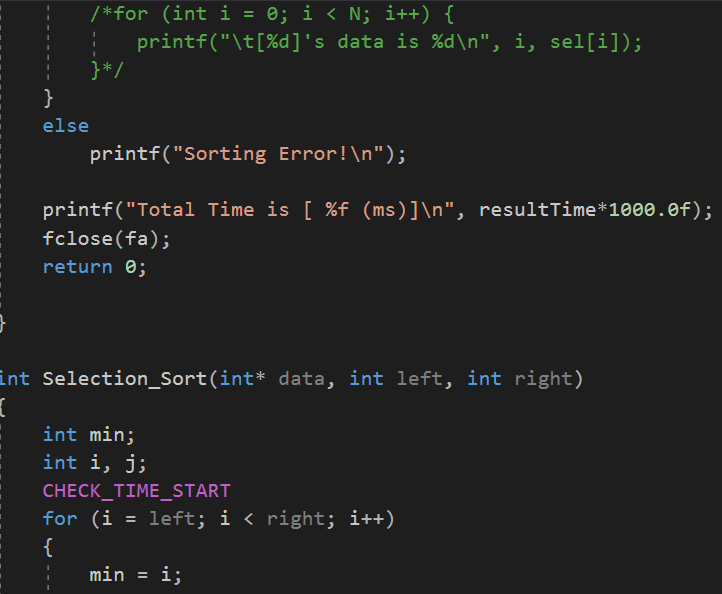


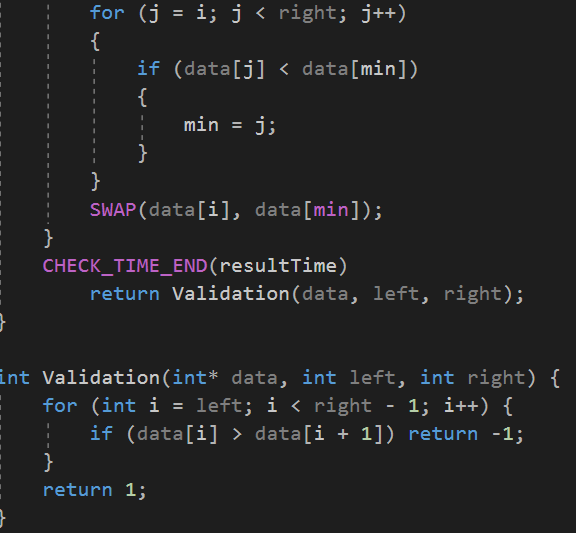


 다음과 같이 구현하였다. 우선 main 함수에서는 파일을 읽어 크기에 맞는 ins 배열을 동적할당하여 데이터를 저장하고 이를 Insertion\_Sort()함수로 넘긴다. Insertion\_Sort함수에서는 인자로 배열과 왼쪽 인덱스, 오른쪽 인덱스를 받아 범위에 맞는 배열을 정렬해준다. 이 때 정렬방법은 n-1번째 까지의 배열을 정렬 되어있다 가정하고, n번째의 데이터의 위치를 n-1번째 까지의 배열을 거슬러 올라가며 알맞은 자리에 삽입하는 insertion sorting 방법을 사용했다. 이 때 시간측정은 전체 for문이 시작할 때 부터 끝날 때 까지를 측정했다. 다 돌리고 난 후에는 Validation 함수를 호출해 정렬이 잘 됐다면 1을 아니라면 -1을 리턴하도록 해주었다. 이 리턴값을 받아 main에서는 정렬이 잘 되었다면 sorting completed라는 문자와 시간을, 아니라면 sorting fail이라는 문자와 시간을 출력한다.

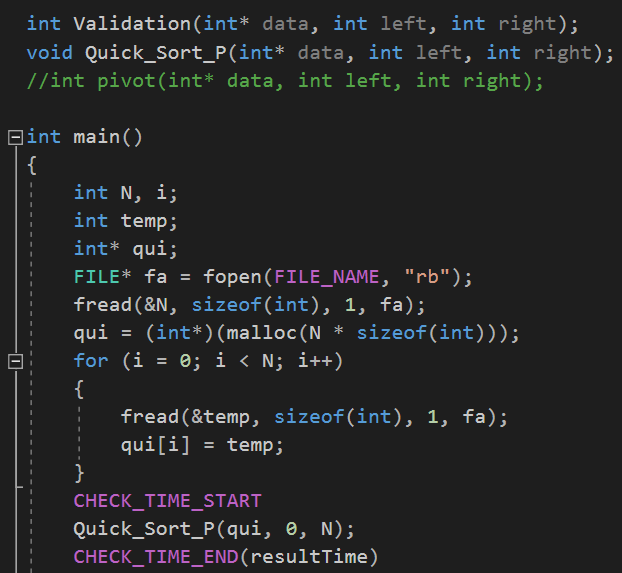
1. Selection Sort

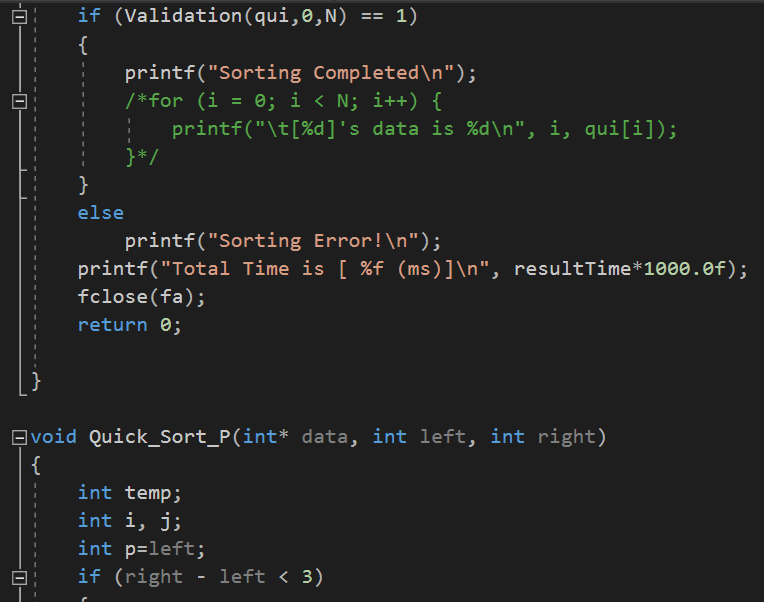


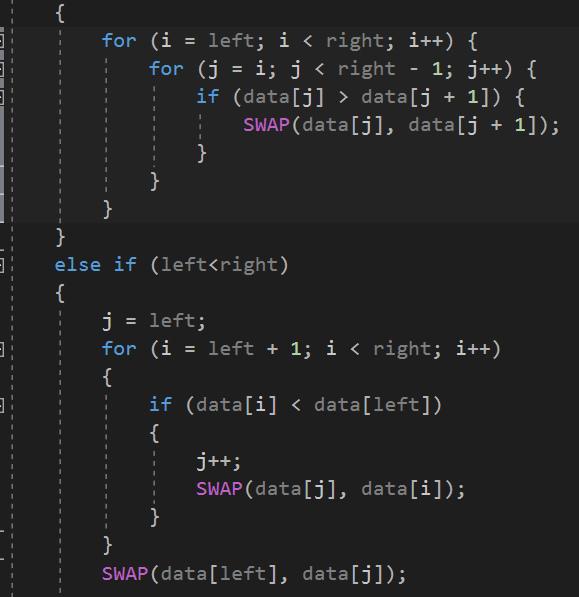


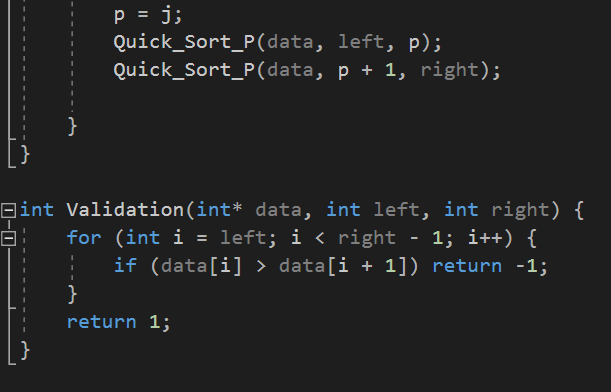
 main함수는 위의 insertion sort 와 같이 배열을 만들어 데이터를 저장하고 Selection\_Sort 함수를 부른다. Selection\_Sort 함수에서는 배열의 범위를 하나씩 좁혀가며 그 중 가장 작은 데이터를 범위의 맨 앞으로 이동시킨다. 이를 위해 min이라는 변수에 최소 원소의 위치를 저장하고 마지막에 범위의 맨 앞 데이터와 min 자리의 데이터를 교환한다. 이 또한 Validation 함수를 마지막에 호출하여 정렬이 잘 되었는지를 판별하고 출력한다. 시간도 전체 for 문이 시작할 때부터 끝날 때 까지를 측정한다.

1. Quick sort P

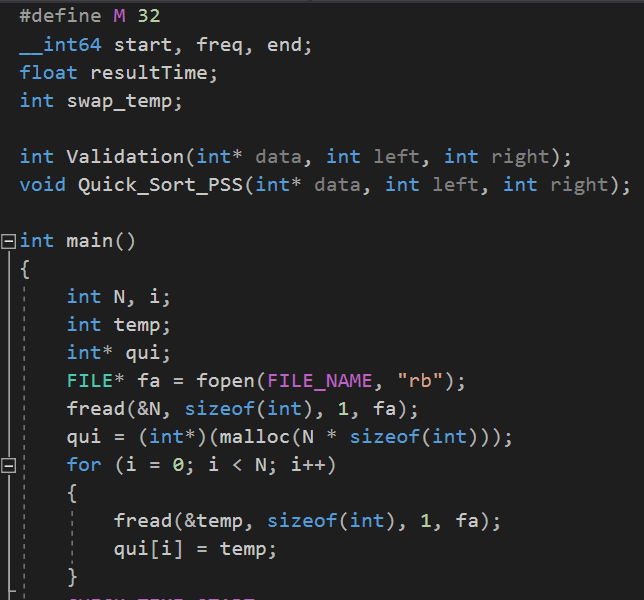


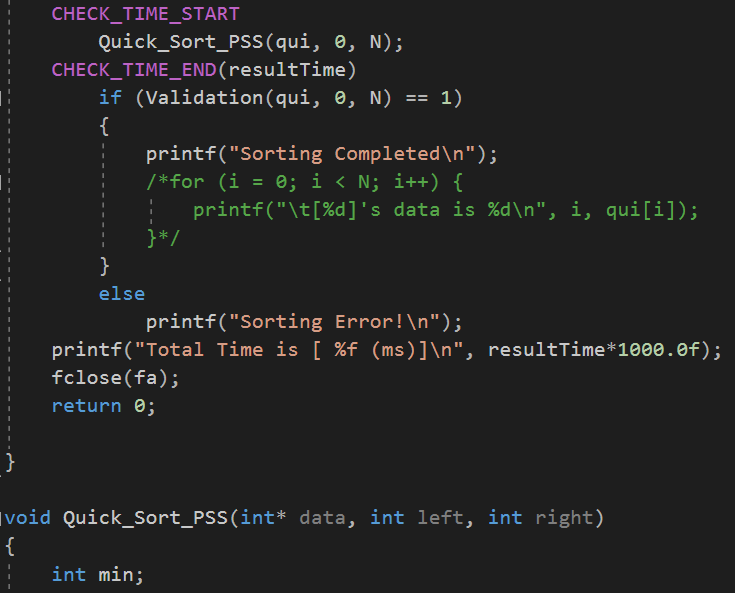


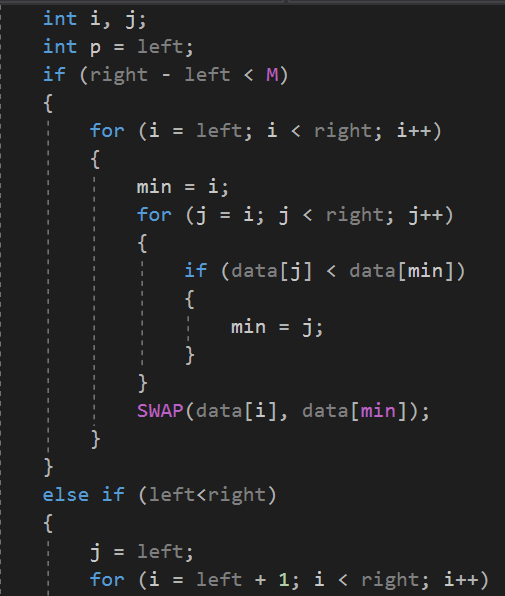


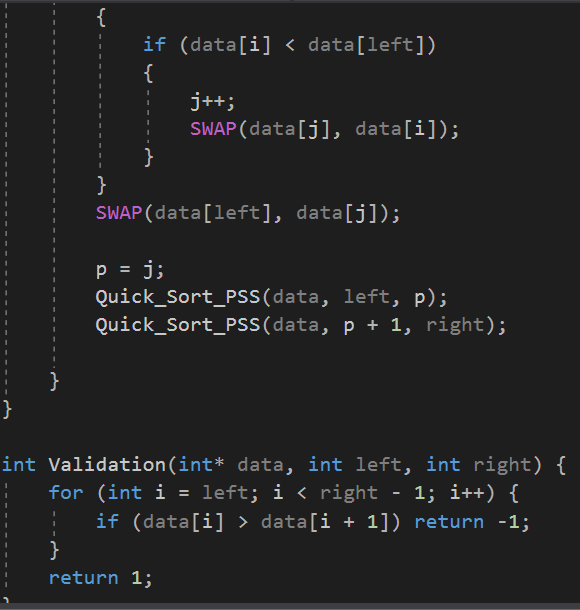
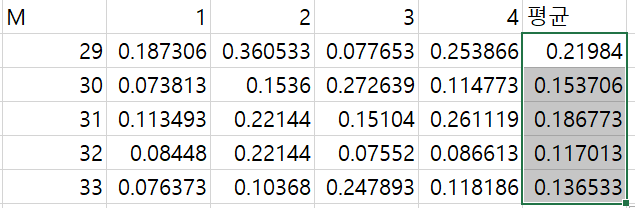
 가장 기본적인 quick sort 함수이다. 우선 main 함수에서는 데이터를 받고 qui 배열에 넣어주며 Quick\_Sort\_P 함수를 불러준다. Quick\_Sort\_P 함수는 void 형 함수로 만약 데이터의 개수가 3개 이하이면 그냥 bubble sort로 정렬해준다. 3개 이상이라면, 가장 왼쪽을 pivot 으로 하여 배열을 pivot보다 작은 쪽과 큰 쪽으로 나눈다. 그 후 작은 쪽을 다시 Quick\_Sort\_P를 불러주고 큰 쪽도 Quick\_Sort\_P를 불러주어 recursive하게 데이터를 정렬해준다. 그 후 main에서 Validation 함수를 불러 올바르게 정렬 되었다면 성공 메시지와 시간을 출력하고 아니라면 실패 메시지와 시간을 출력해준다. 이 때 시간측정은 위의 두 방법과는 다르게 recursive한 방법이므로 main에서 Quick\_Sort\_P를 호출하기 전부터 호출 이후까지를 측정한다.

1. Quick sort PSS

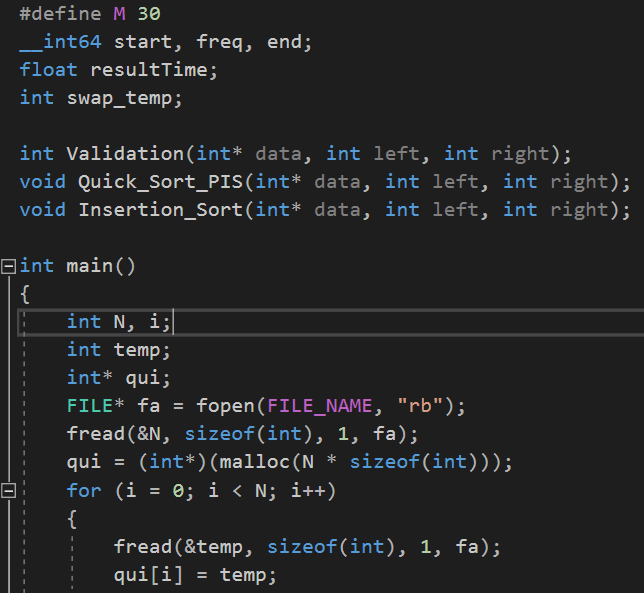


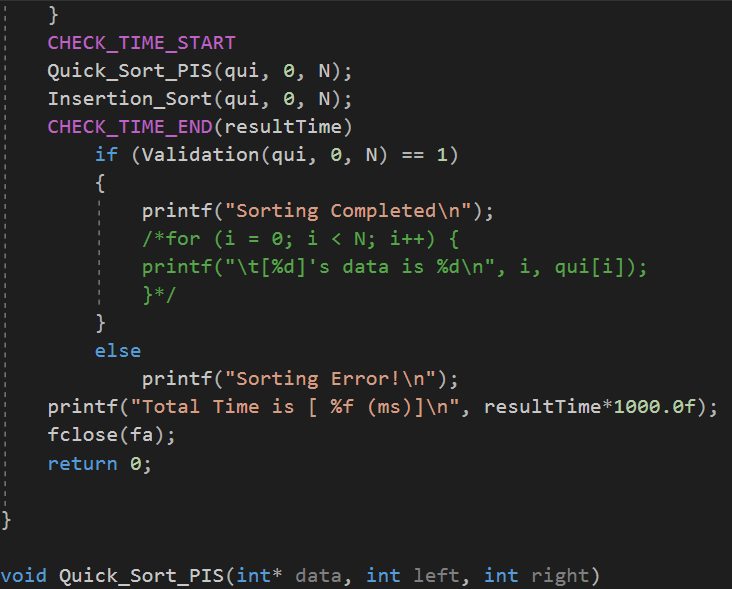


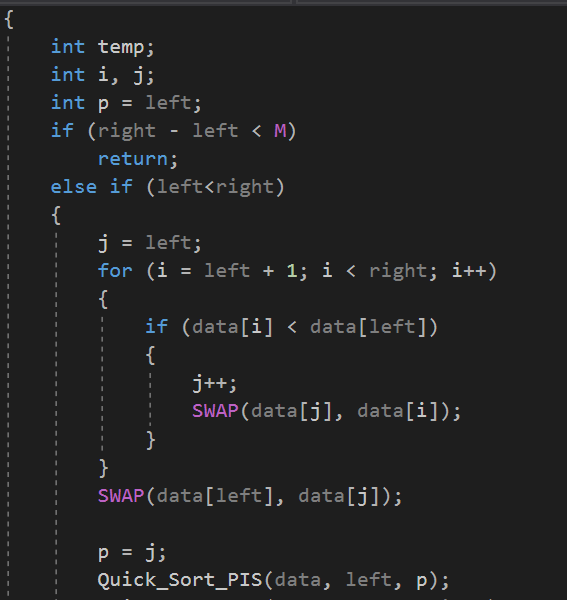


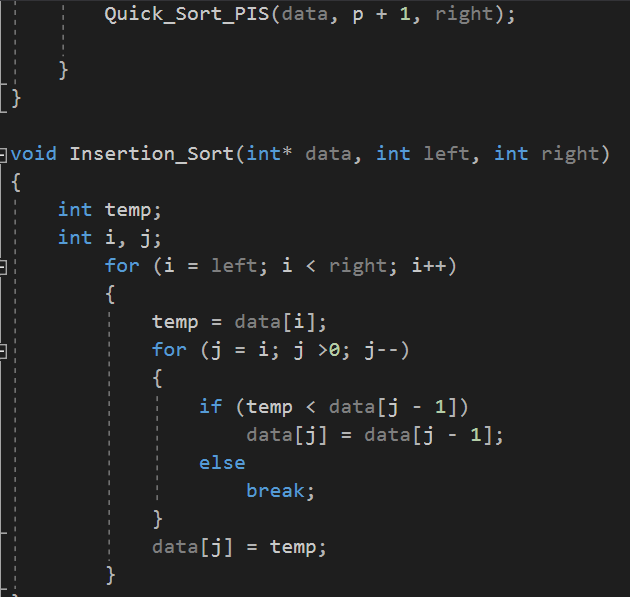
 기본적인 Quick sort 방식에 만약 크기가 M 이하일 경우 selection sort를 통해 정렬하도록 한 sorting 방법이다. 모든 부분은 위의 Quick sort와 동일하며, right-left<M일 경우의 부분만 selection sort로 정렬한다. 이 때 M은  이 실험을 통해 32로 설정하였다.

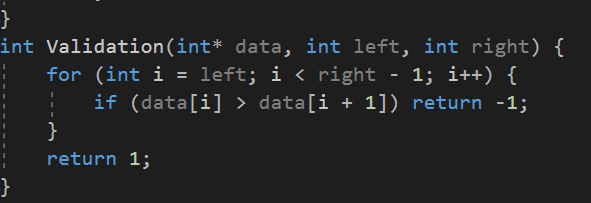
1. Quick sort PIS

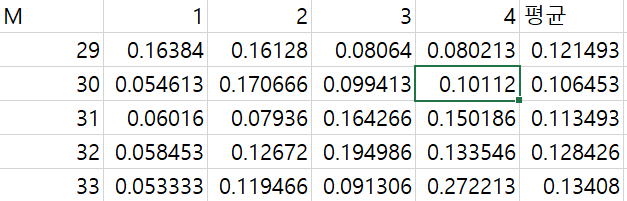




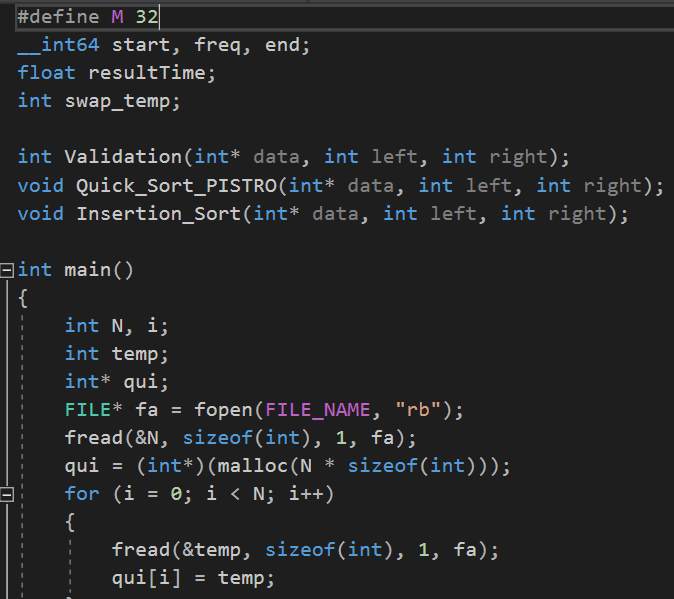


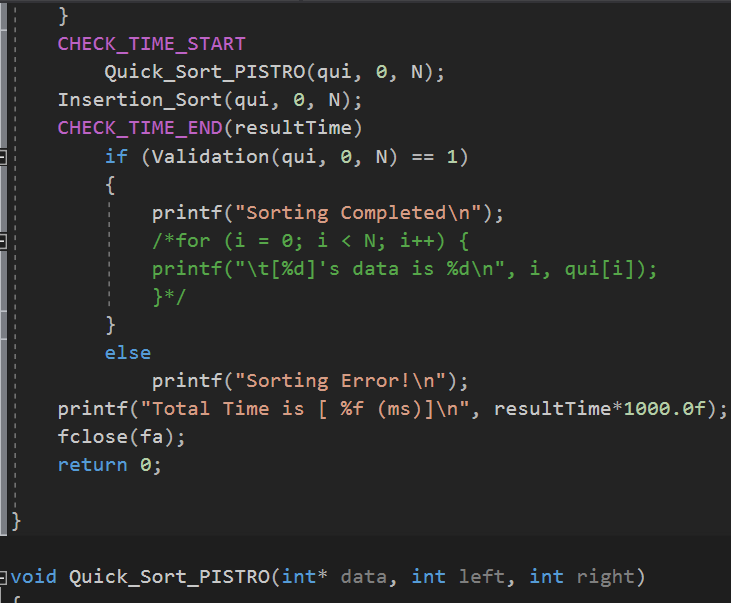


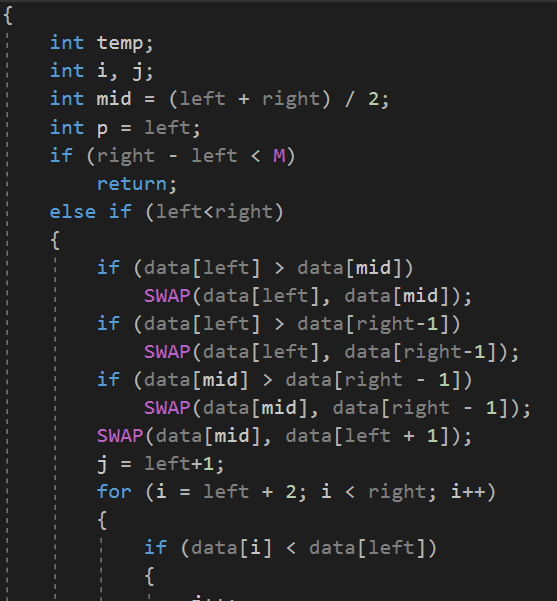
 기본적인 quick sort 방법에 insertion sort 를 추가한 방법이다. 먼저 quick sort 처럼 main 에서는 데이터를 받고 배열을 만들어 먼저 Quick\_Sort\_PIS함수를 불러준다. Quick\_Sort\_PIS 함수에서는 만약 크기가 M보다 작다면 아무것도 하지 않고 그냥 return 해준다. M보다 클 경우에는 기본적인 Quick sort를 돌려 재귀적으로 정렬한다. 모든 재귀가 끝난 뒤 main에서는 다시한번 Insertion Sort 함수를 불러 M 이하의 개수에서 정렬되지 않은 부분들을 정렬해준다. 이 때 M은 실험을 통해

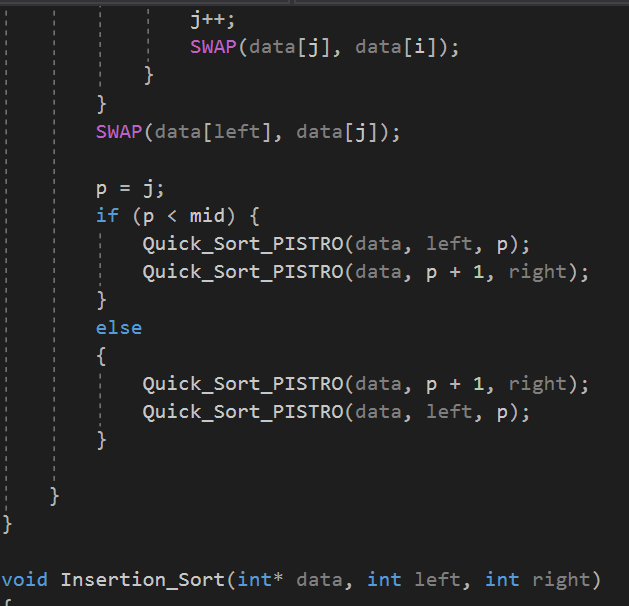
 30 으로 결정하였다. 이후 Validation 함수를 불러 정렬이 성공적으로 되었는지에 따라 메시지와 시간을 출력한다. 이 때 시간측정은 main에서 Quick sort와 Insertion sort를 모두 호출하여 정렬이 끝날 때 까지로 호출전부터 호출 이후 까지를 측정한다.

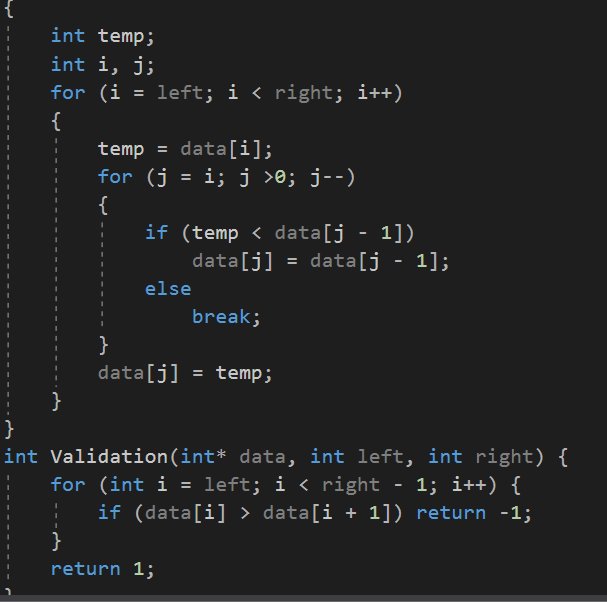
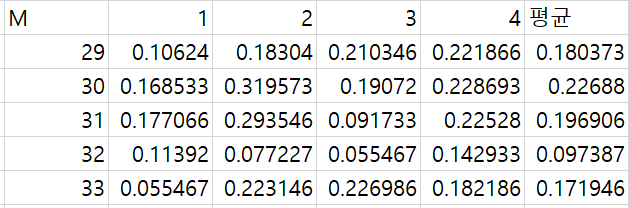
1. Quick sort PISTRO









 다양한 improvement를 적용한 quick sort 이다. 먼저 main에서는 데이터를 받고 Quick\_Sort\_PISTRO를 호출한다. Quick\_Sort\_PISTRO에서는 크기가 M이하일 경우 바로 리턴하여 정렬하지 않는 위의 PIS 방법을 사용한다. 크기가 M 보다 클 경우에는 Quick sort을 하는데 이 때 pivot 을 위와는 다른 방식을 사용한다. 먼저 데이터의 중간과 처음, 끝의 크기를 비교하여 정렬하고 중간 크기의 값을 left+1의 자리에 놓는다. 이렇게 하면 중간크기의 값이 중간에 올 확률이 늘어나 효율이 조금 더 좋아진다. 그 후 left+1, 즉 중간크기의 값을 pivot으로 하여 데이터를 큰 쪽과 작은 쪽으로 나눈다. 다 나눈 후 pivot 의 위치에 따라 큰 쪽과 작은 쪽 중 크기가 더 작은 쪽을 먼저 재귀로 돌리는 tail recurtion 방식을 사용한다. 이를 위해 pivot 의 위치가 중간보다 앞 쪽이면 작은 쪽이 더 적으므로 이쪽부터 Quick\_Sort\_PISTRO를 부르고 그 후 큰 쪽의 Quick\_Sort\_Pistro를 불러준다. 만약 반대로 pivot의 위치가 중간보다 뒤라면, 큰 쪽의 크기가 더 작으므로 큰 쪽부터 재귀를 돌리고 그 후 작은 쪽을 돌린다. 이를 모두 마친 후 M 보다 작은 경우가 정렬이 안됐으므로 main에서 Insertion sort 를 불러 나머지를 정렬해준다. 이 때 M은  실험 결과에 따라 32로 확정했다. 이 후 Validation 함수를 불러 메시지와 시간을 출력한다. 시간측정은 위와 같다.

**실험**

우선 원소의 개수 n은 데이터를 정렬하는데 딜레이가 현실적인 시간일 때에만 생각하였다. 이 경우 quick sort는 최대 2^20까지 가능하였고 최대 원소 2^20과 일반적인 경우인 2^10개를 비교하였다. Insertion과 Selection은 2^20을 넣었을 경우 현실적인 시간 내에 종료되지 않아 그보다 더 작은 2^15 일 경우를 실험하였다.

1. Entirely Random
2. Insertion Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.713385 | 0.673279 | 0.672425 | 1.345277 | 0.966825 | 0.874238 |
| 2^15 | 158.775116 | 167.002533 | 173.666199 | 160.527435 | 155.026871 | 162.9996 |

1. Selection Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 1.622610 | 1.508690 | 1.532583 | 1.728850 | 2.133329 | 1.705212 |
| 2^15 | 1604.799561 | 1603.623291 | 1605.228271 | 1606.370605 | 1597.652466 | 1603.535 |

1. Quick Sort P

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.121173 | 0.098560 | 0.085333 | 0.115626 | 0.090880 | 0.102314 |
| 2^20 | 120.511314 | 119.980118 | 121.351845 | 126.744896 | 121.856171 | 122.0889 |

1. Quick Sort PSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.073387 | 0.196693 | 0.066987 | 0.178773 | 0.066987 | 0.116565 |
| 2^20 | 126.855400 | 127.716858 | 127.596092 | 130.540085 | 126.387787 | 127.8192 |

1. Quick Sort PIS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.073813 | 0.066987 | 0.101546 | 0.050347 | 0.075947 | 0.073728 |
| 2^20 | 108.966164 | 108.295876 | 108.743454 | 110.291817 | 108.739182 | 109.0073 |

1. Quick Sort PISTRO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.097706 | 0.082346 | 0.050773 | 0.081920 | 0.065280 | 0.075605 |
| 2^20 | 114.664719 | 118.342567 | 116.092339 | 117.986740 | 116.977669 | 116.8128 |

1. Ascending
2. Insertion Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.001707 | 0.003413 | 0.001707 | 0.001707 | 0.001280 | 0.001963 |
| 2^15 | 0.071253 | 0.038400 | 0.097280 | 0.086186 | 0.085333 | 0.07569 |

1. Selection Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 2.406395 | 1.512530 | 1.480530 | 1.916156 | 1.485650 | 1.760252 |
| 2^15 | 1606.970459 | 1603.397461 | 1601.492920 | 1602.031860 | 1607.573730 | 1604.293 |

1. Quick Sort P (딜레이로 인해 2^15로 측정함)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.573865 | 1.532157 | 0.925865 | 1.519783 | 1.314984 | 1.173331 |
| 2^15 | 507.424652 | 519.378540 | 512.309082 | 504.022827 | 549.379517 | 518.5029 |

1. Quick Sort PSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 1.288957 | 0.555519 | 1.169064 | 0.557652 | 0.747092 | 0.863657 |
| 2^15 | 512.706787 | 507.935333 | 503.621338 | 507.984802 | 520.112000 | 510.4721 |

1. Quick Sort PIS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.732585 | 0.848638 | 0.922451 | 0.457386 | 0.951891 | 0.78259 |
| 2^15 | 520.719116 | 509.206360 | 508.383759 | 494.921631 | 510.197510 | 508.6857 |

1. Quick Sort PISTRO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.179200 | 0.11093 | 0.190293 | 0.055893 | 0.055040 | 0.118271 |
| 2^15 | 20.811049 | 8.669848 | 10.987069 | 12.423654 | 9.066220 | 12.39157 |

1. Descending
2. Insertion Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.371199 | 0.168960 | 0.147626 | 0.253439 | 0.148053 | 0.217855 |
| 2^15 | 155.972366 | 151.084900 | 152.654190 | 156.598282 | 154.398819 | 154.1417 |

1. Selection Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 1.956689 | 2.970020 | 1.665703 | 2.252795 | 2.022822 | 2.173606 |
| 2^15 | 1623.368896 | 1618.766479 | 1620.019165 | 1624.833740 | 1617.602417 | 1620.918 |

1. Quick Sort P

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.070827 | 0.107946 | 0.046933 | 0.074667 | 0.046507 | 0.069376 |
| 2^15 | 16.737671 | 22.331259 | 16.921562 | 22.249340 | 28.233753 | 21.29472 |

1. Quick Sort PSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.047360 | 0.060160 | 0.162560 | 0.109226 | 0.066133 | 0.089088 |
| 2^15 | 19.357824 | 18.890200 | 23.677816 | 16.099379 | 21.855099 | 19.97606 |

1. Quick Sort PIS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.040107 | 0.053333 | 0.042667 | 0.040533 | 0.023893 | 0.040107 |
| 2^15 | 18.111534 | 21.256487 | 17.407110 | 17.617027 | 20.009771 | 18.88039 |

1. Quick Sort PISTRO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.088746 | 0.078506 | 0.146773 | 0.058453 | 0.044373 | 0.08337 |
| 2^15 | 10.734910 | 16.007645 | 10.954643 | 19.402199 | 14.259597 | 14.2718 |

1. Few swaps
2. Insertion Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.010240 | 0.013227 | 0.008107 | 0.00853 | 0.02304 | 0.012629 |
| 2^15 | 0.276053 | 0.263679 | 0.256426 | 0.128426 | 0.097706 | 0.204458 |

1. Selection Sort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 1.532583 | 4.563190 | 2.409808 | 4.771830 | 1.896529 | 3.034788 |
| 2^15 | 1706.083984 | 1705.520874 | 1699.182739 | 1675.038086 | 1690.291016 | 1695.223 |

1. Quick Sort P

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.833705 | 1.275731 | 1.088424 | 1.203624 | 0.718505 | 1.023998 |
| 2^15 | 390.795532 | 404.014496 | 347.548706 | 535.175781 | 534.357910 | 442.3785 |

1. Quick Sort PSS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.763732 | 1.00821 | 0.971945 | 1.023571 | 1.733970 | 1.100286 |
| 2^15 | 339.881500 | 429.020111 | 432.715454 | 332.235260 | 406.969147 | 388.1643 |

1. Quick Sort PIS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.929278 | 1.241597 | 0.793598 | 0.476586 | 0.986025 | 0.885417 |
| 2^15 | 422.993866 | 342.976135 | 385.428070 | 390.447388 | 410.446503 | 390.4584 |

1. Quick Sort PISTRO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 횟수  N | 1 | 2 | 3 | 4 | 5 | 평균시간 |
| 2^10 | 0.109653 | 0.180480 | 0.179626 | 0.210773 | 0.063147 | 0.148736 |
| 2^15 | 14.326583 | 35.000671 | 27.063410 | 34.930275 | 32.568249 | 28.77784 |

**분석**

1. Insertion Sort : 삽입 정렬의 경우 데이터의 상태에 따라 시간의 차이가 심하다. 먼저 일반적인 랜덤으로 데이터가 만들어질 경우는 시간복잡도가 O(n^2/4)로 그에 맞는 시간이 측정된다. 하지만 이 또한 랜덤으로 만들어지기 때문에 안의 정렬 상태에 따라 시간이 천차만별로 측정된다. 만약 ascending 한 경우에는 최선의 경우로, insertion sort가 linear time에 정렬이 이루어진다. 그러므로 시간을 보면 다른 경우와 비교했을 때 월등하게 적은 시간을 측정할 수 있다. Descending 한 경우는 최악의 경우로 시간복잡도가 O(n^2/2)이다. 정렬을 할 때 모든 수를 다시 전부 검사하며 자리를 찾아야 하기 때문이다. 그로인해 시간이 다시 크게 증가하였다. Few swap을 한 경우에는 ascending 경우보다는 시간이 오래 걸리지만, 이 또한 자리를 찾는데 시간이 적게 걸리고 거의 linear 하게 측정된다. 따라서 이 경우에도 굉장히 적은 시간이 걸렸다.
2. Selection Sort : 선택정렬은 데이터의 상태를 고려하지 않는다. 무조건 이중루프를 돌려 모든 숫자를 검사하고 가장 작은 것을 앞에 위치시키므로, 데이터가 정렬이 되었든, 랜덤하게 되어있든 상관없이 O(n^2) 만큼의 시간이 걸린다. 따라서 모든 경우의 시간이 가장 오래 걸리고, 어떤 경우든 시간의 차이가 별로 나지 않는다.
3. Quick Sort P : 일반적인 퀵소트로 평범한 랜덤 데이터를 돌릴 때의 시간복잡도는 O(nlogn) 이다. 이는 위의 두 방법과 비교해서 매우 적은 시간으로 그에 따라 시간 측정을 보면 데이터의 개수가 늘어나도 훨씬 짧은 시간에 정렬이 가능하다. 하지만 이는 Ascending 할때에 최악의 시간 복잡도를 가지게 되는데, 왜냐하면 맨 왼쪽을 pivot으로 구현한 경우 재귀가 T(n)=T(n-1) + c 로 재귀했을 때의 이득이 전혀 없고 오히려 시간복잡도가 O(n^2)에 가깝게 된다. 그러므로 매우 느리게 정렬이 이루어지고 랜덤 데이터로 2^20 까지 돌릴 수 있었던 것에 반해 2^15 정도까지 밖에 돌리지 못한다. 반대로 descending의 경우 시간이 랜덤보다는 느리지만 ascending 보다는 매우 빨라지는데 이유는 pivot을 구할 때 데이터의 이동이 이루어지며 위의 ascending 한 경우보다 나누어지는 부분이 더 많아 재귀가 제 일을 하기 때문이다. Few swaps 의 경우에도 오로지 T(n-1)로 나누어질 때 보다 훨씬 좋게 나누어지기 때문에 어느정도 현실적인 시간 내에 시간 측정을 할 수 있다.
4. Quick Sort PSS : 위의 일반적인 퀵소트에 selection sort를 더한 것으로 거의 모든경우에 위의 퀵소트와 경향성이 비슷하다. 하지만 모든 경우에서 Quick sort P보다 조금 시간이 덜 걸리는 모습을 보여주는데 이 이유는 위에서는 M보다 작을 때 버블소트를 돌리지만 이 때에는 버블보다 조금 빠른 선택정렬을 돌리기 때문으로 보인다.
5. Quick Sort PIS : 이 또한 일반적인 퀵소트와 모든 경향성이 비슷하다. 하지만 이 방식은 또한 Quick sort P, Quick sort PSS 보다 조금 더 나은 시간을 보이는데, 이는 M보다 작을 때 아예 정렬을 하지 않고, 맨 마지막에 정렬이 별로 남지 않았을 때 시간복잡도가 Linear time에 가까운 Insertion sort를 돌리기 때문인 것으로 보인다.
6. Quick Sort PISTRO : 모든 경우에서 거의 최적에 가까운 시간을 보여준다. 먼저 random으로 데이터를 만든 경우에는 위의 퀵소트들과 시간적인 측면에서 별로 차이가 나지 않지만, ascending, descending에 들어서면 이 방식의 장점이 드러난다. 이 방식에서 시간을 매우 짧게 줄여주는데 가장 큰 공헌을 하는 방식은 바로 midian을 구하는 방식이다. 이 방식에서는 처음, 끝, 중간 중에 중간값을 pivot으로 선택하기 때문에 ascending이나 descending의 경우 확정적으로 pivot이 가장 중간에 위치하게 된다. 그러므로 T(n)=2T(n/2) + c 가 되어 시간복잡도가 크게 줄어든다. 때문에 어떤 데이터를 넣어도 별다른 딜레이 없이 정렬된 값을 구할 수 있다.

**질문**

1. 수행 시간과 이 방법들의 이론적인 시간 복잡도간에 어떤 연관성을 발견할 수 있는가?

시간 복잡도에 N을 실험한 시간에 따라 넣어보면, 컴퓨터 간의 차이로 오차는 있을지 몰라도, 수행 시간이 비슷하거나, 혹은 N을 증가시켰을 때의 증가량이 비슷하다.

1. Entirely random 데이터의 크기를 변화시켜가면서 실험을 하였을 때, Insertion sort와 Selection sort 간에 두드러진 속도 차이를 느낄 수 있는가?

느낄 수 있다. Selection sort의 경우 모든 데이터를 검사하기 때문에 확정적으로 n^2의 시간이 걸리지만, insertion sort의 경우 데이터가 랜덤이라도 모든 데이터를 검사하지 않고 그저 자리를 찾으면 다음 루프로 넘어가기 때문에 시간이 훨씬 줄어든다.

1. Entirely random 데이터의 크기를 변화시켜가면서 실험을 하였을 때, Quick sort P 방법 대비 Insertion sort 방법의 수행 시간에 대하여 n/ logn와 같은 비율이 측정되었는가?

어느 정도의 오차가 있지만, 그에 준하는 비율의 시간차이가 생겼다.

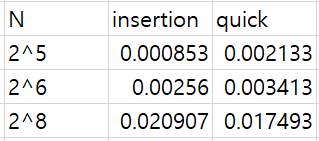
1. 과연 Quick sort PSS (Pivot+SelectionSort) 방법이 다른 방법에 비하여 의미가 있는가? – 이때 서로 다른 종류의 데이터에 대하여 수행 시간에 차이가 있는가? 있다면 그 이유는 무엇으로 추정되는가?

안을 본인처럼 bubble sort로 구현하였다면 의미가 있다. 하지만 다른 퀵소트랑 비교해본다면, 오히려 안좋은 방식의 퀵소트이다. Selection sort 자체가 시간복잡도가 높은 별로 좋지 않은 sorting 방식이기 때문이다. 수행시간의 차이는 위에서 기술한 대로 pivot을 구하는 방식에 의해 차이가 나는 것으로 selection sort 자체가 시간을 줄이거나 하지는 않는 것 같다.

1. 과연 어떤 부류의 데이터에 대해 insertion sort 방법의 수행 시간이 quick sort 방법에 비해 그리 느리지 않은가?

정렬된 경우나 얼추 정렬된 데이터일 경우이다. 이 때에는 수행 시간이 linear time으로 매우 짧아지기 때문이다.

1. n 값이 작을 경우 굳이 quick sort 방법을 사용할 필요없이 insertion sort 방법을 사용해도 크게 문제가 되지 않는다. 과연 이는 어느 정도까지의 n 값까지일 지 실험을 통하여 그 값을 밝혀라

로 약 2^8 까지 이다.

1. 각 방법을 최대한 효율적으로 구현하기 위하여 어떤 노력을 기울였는가?

매크로를 최대한으로 활용하였고, PISTRO의 경우 할 수 있는 모든 improvement를 사용하였다.