**알고리즘 설계와 분석 HW1 보고서**

* **효율적인 정렬 방법의 구현**

학번: 20181688

이름 : 조태연

1. **실험 환경**

OS : Windows 10 Education

Cpu : Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz

Ram : 16.0GB

Compiler : Visual Studio 2022 Release Mod

1. **수행시간 Table(단위 ms)**
2. **INSERTION\_SORT**

|  |  |  |  |
| --- | --- | --- | --- |
| Size \  Data format | Entirely random | Descending | Few swaps |
| 1024 | 2.86 | 5.8725 | 0.297900 |
| 4096 | 46.07 | 93.659897 | 1.865100 |
| 16384 | 751.393494 | 1481.114136 | 14.3416 |
| 65536 | 11852.412109 | 23533.111328 | 126.185699 |
| 262144 | 191566.343750 | 408408.34596 | 975.137024 |
| 1048576 | x | x | 8294.714844 |

1. **HEAP\_SORT**

|  |  |  |  |
| --- | --- | --- | --- |
| Size \  Data format | Entirely random | Descending | Few swaps |
| 1024 | 0.706800 | 0.958200 | 1.712000 |
| 4096 | 4.325300 | 3.454700 | 4.376900 |
| 16384 | 36.452900 | 45.042900 | 43.103500 |
| 65536 | 788.153625 | 657.783386 | 735.604370 |
| 262144 | 13153.453125 | 11786.936523 | 13539.451172 |
| 1048576 | 277905.539204 | 256824.145329 | 303821.421892 |

1. **QUICK\_SORT**

|  |  |  |  |
| --- | --- | --- | --- |
| Size \  Data format | Entirely random | Descending | Few swaps |
| 1024 | 0.193200 | 0.060200 | 0.050500 |
| 4096 | 0.838200 | 0.241600 | 0.199100 |
| 16384 | 3.909000 | 1.008600 | 1.073000 |
| 65536 | 15.157400 | 4.147000 | 3.695800 |
| 262144 | 77.891800 | 21.726299 | 16.769300 |
| 1048576 | 325.667389 | 87.773697 | 69.107399 |

1. **OPTIMIZED\_QUICK\_SORT**

|  |  |  |  |
| --- | --- | --- | --- |
| Size \  Data format | Entirely random | Descending | Few swaps |
| 1024 | 0.161000 | 0.054000 | 0.055400 |
| 4096 | 0.733500 | 0.226300 | 0.266700 |
| 16384 | 3.453700 | 0.9579001 | 0.912700 |
| 65536 | 14.875400 | 3.946300 | 3.333500 |
| 262144 | 68.846703 | 18.943701 | 17.559299 |
| 1048576 | 320.908813 | 81.795403 | 75.408699 |

1. **데이터 분석**

**-서로 다른 종류의 데이터별로 수행 시간의 차이가 있는가?**

**:** Entirely random과 Descending 데이터의 경우에는 4가지 방법의 우열이 똑같은데 Few swaps 데이터의 경우에 Heap sort가 insertion sort보다 느린 결과가 나왔다.

**-quick sort는 이름이 의미하는 바와 같이 정말 빠른가?**

**:** 데이터 테이블을 보면 Insertion과 Heap에 비해 확실히 빠른 결과가 나왔다**.**

**-어떤 부류의 데이터에 대해 insertion sort 방법의 수행 시간이 heap sort 또는 quick sort 방법에 비해 빠르거나 그리 느리지 않은가? 과연 insertion sort 방법이 거의 선형적인 시간에 작동하는 경우를 발견하였는가?**

**:** Few swap 데이터에서 insertion sort가 heap sort보다 빠른 결과가 나왔다. 밑에 4번 항목에 있는 데이터를 보면 10<=n<=25에서 선형적인 시간에 작동하는 경우를 볼 수 있다.

**4. 더 자세한 실험**

**-n값이 작을 경우 굳이 quick sort방법을 사용할 필요없이 insertion sort방법을 사용해도 크게 문제가 되지 않는다. 과연 이는 어느 정도 범위의 n값까지 일지 Entirely random 데이터에 대한 실험을 통하여 자신의 값을 제시하라**

|  |  |  |
| --- | --- | --- |
| Size \  Sorting method | **INSERTION\_SORT** | **QUICK\_SORT** |
| **10** | 0.000900 | 0.001500 |
| **20** | 0.002100 | 0.003700 |
| **25** | 0.002500 | 0.003800 |
| **30** | 0.010500 | 0.004400 |

**:** 25<n<30 사이에서 QUICK\_SORT가 INSERTION\_SORT보다 빨라지는 지점이 있습니다.

그 전까지는 오히려 INSERTION\_SORT가 QUICK\_SORT보다 빠른 모습을 보입니다.

Sort 알고리즘이 쓰이는 프로그램의 유형에 따라 다르겠지만 자신의 값을 제시한다고 생각했을 때 제 기준에서 10초정도까지는 답답함을 느끼지 않기 때문에 n<65536 범위 내에서는 INSERTION\_SORT와 QUICK\_SORT을 구분없이 사용해도 된다고 생각합니다.

**-** **입력 데이터의 크기 n에 quick sort 방법을 수행할 때 재귀적으로 생성되는**

**트리의 깊이가 어느 정도 되는지 각 리프 노드의 레벨에 대한 최소/최대/평**

**균을 적절히 크기가 큰 Entirely random 데이터에 대한 실험을 통하여**

**자신의 값을 제시하라. 평균적으로 최악의 경우에 해당하는 O(n)에 가까운**

**지 아니면 수업 시간에 이론적으로 증명한 바와 같이 O(n log n)에 가까운**

**지 실험적으로 분석할 것**

**:** 262144개의 크기를 기준으로 6세트의 난수 세트를 만들어서 QUICK\_SORT시 생기는 트

리의 깊이를 구해봤습니다. 그 결과 각각 트리의 깊이는 236217, 235796, 235995, 235981, 235899, 236235로 최소/최대/평균은 235796/236235/236,020.5입니다.

2-3의 테이블로 확인해보면 n의 개수가 4배씩 늘고 있을 때 시간이 4~5배 사이로 늘어나

는 것을 확인할 수 있다. 이를 미루어보아 O(n^2)보다는 O(nlogn)에 가깝다고 볼 수 있다.