CSE3018: 알고리즘 설계와 분석

**HW2: Quick Sort 방법의 효율적인 구현**

2023년 4월 19일

**1. Code 구현 내용**

**(1) Insertion Sort**

구현 결과

구현이 성공적으로 이루어졌는지 알아보기 위해 sample data를 이용해 테스트해 보았다. 사진은 캡처를 용이하기 하기 위해 32개의 data를 사용했으며 그 이상의 데이터에 대해서도 모두 sorting이 잘 된 것을 확인할 수 있었다.

Text

Description automatically generated

**(2) Merge Sort**

구현 결과

Text

Description automatically generated

**(3) Quick Sort NAÏVE**

구현 결과

Text

Description automatically generated

**(4) Quick Sort P**

Text

Description automatically generated

**(5) Quick sort PIS**

**Text

Description automatically generated**

**(6) Quick Sort PISTRO**

**Text

Description automatically generated**

**2. 수행 결과**

(1) Insertion Sort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **rd** | **as** | **de** | **fw** |
| **32 (2^5)** | 0.006 | 0.005 ms | 0.010ms | 0.005ms |
| **1024 (2^10)** | 0.108 | 0.154 | 0.055 | 0.025 |
| **16384 (2^14)** | 0.384 | 0.204 | 0.803 | 0.518 |
| **2^20** | 6.868 | 34.430 | 56.327 | 20.671 |
| **2^21** | 34.183 | 59.632 | 836.882 | 105.127 |

(2) Merge Sort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **rd** | **as** | **de** | **fw** |
| **32 (2^5)** | 0.009 | 0.009 | 0.010 | 0.010 |
| **1024 (2^10)** | 0.098 | 0.104 | 0.095 | 0.093 |
| **16384 (2^14)** | 1.499 | 1.675 | 1.495 | 5.499 |
| **2^20** | 107.530 | 110.060 | 107.175 | 106.480 |

(3) Quick Sort NAIVE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **rd** | **as** | **de** | **fw** |
| **32 (2^5)** | 0.001 | 0.001 | 0.005 | 0.001 |
| **1024 (2^10)** | 0.249 | 0.189 | 0.328 | 0.361 |
| **16384 (2^14)** | 113.997 | 114.36 | 131.151 | 115.263 |
| **2^20** | overflow | overflow | overflow | overflow |
| **2^17** | 7560.062 | 7567.143 | 7563.121 | 7592.072 |

(4)Quick Sort with various strategy

Rd 데이터를 이용하여 측정하였다.

|  |  |
| --- | --- |
|  | **QS\_P** |
| **1024 (2^10)** | 0.236 |
| **4096 (2^12)** | 4.466 |
| **16384 (2^14)** | 80.713 |

-PIS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **PIS**  M=10 | **PIS**  M=100 | **PISTRO**  M = 10 | **PISTRO**  M = 100 |
| **1024 (2^10)** | 0.247 | 1.258 | 0.283 | 1.308 |
| **4096 (2^12)** | 6.646 | 29.434 | 6.121 | 6.111 |
| **16384 (2^14)** | 478.505 | 478.423 | 104.949 | 104.442 |

모든 단위는 ms로 측정되었다.

**3. 분석**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **rd** | **as** | **de** | **fw** |
| **32 (2^5)** | 0.006 | 0.005 ms | 0.010 | 0.005 |
| **1024 (2^10)** | 0.108 | 0.154 | 0.055 | 0.025 |
| **16384 (2^14)** | 0.384 | 0.204 | 0.803 | 0.518 |
| **2^20** | 6.868 | 34.430 | 56.327 | 20.671 |
| **2^21** | 34.183 | 59.632 | 836.882 | 105.127 |

Data 유형에 따른 시간 차이

Insertion Sort의 수행 결과를 보면 Few swaps 데이터와 descending 간의 정렬 시간이 크게 차이가 나는 점을 확인할 수 있다. 이는 insertion sort의 이론적인 내용과 같은 결과임을 볼 수 있다. Decsending data는 다른 타입의 데이터와 비교해서도 시간이 더 오래 걸리는 것을 확인할 수 있다.

Quick Sort strategy간 시간 차이 – pivot 유무에 따라

동일한 rd 데이터로 Quick Sort Naïve와 Quick Sort Pivot을 실험한 결과가 다음과 같이 나왔다. Naïve의 경우 특별한 연산 없이 left의 원소를 pivot으로 설정하였으며 Quick Sort Pivot에서는 몇 가지 방법으로 pivot을 다르게 설정하여 시험해보았으며, stack over flow와 연산에서의 추가 비용을 고려했을 때, Median of 3 방법이 효과를 가진다는 것을 확인하였다.

|  |  |  |
| --- | --- | --- |
|  | **Navie** | **QS\_P 1** |
| **16384 (2^14)** | 113.978 | 80.615 |
| **1024 (2^10)** | 0.249 | 0.987 |
| **8192 (2 ^13)** | 26.723 | 0.204 |
| **131072(2^17)** | 7560.062 | 5332.146 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **PIS**  M=10 | **PIS**  M=100 | **PISTRO**  M = 10 | **PISTRO**  M = 100 |
| **1024 (2^10)** | 0.247 | 1.258 | 0.283 | 1.308 |
| **4096 (2^12)** | 6.646 | 29.434 | 6.121 | 6.111 |
| **16384 (2^14)** | 478.505 | 478.423 | 104.949 | 104.442 |

위의 결과를 보면 insertion sort가 적용되는 기준인 M의 범위에 따라 차이를 가진다는 점을 볼 수 있다. Data의 사이즈가 1024 일때 M = 100이면 전체 데이터의 10프로 정도를 기준으로 설정한 것인데, 이때의 insertion sort는 효율을 높이는데 좋지 않아보인다.

이론적으로 Insertion Sort보다 Quick Sort가 더 빠른 시간에 수행되어야 하나, 측정 결과 Insertion Sort가 더 빠르게 시행되었다. Insertion Sort를 함수 구현 과정에서 사용한 연산 등이 달라 메모리 할당 등의 문제로 인해 발생한 오류인 것 같다. 같은 자료구조를 사용하여 구현해야 보다 정확한 비교를 할 수 있었을 것인데, 알고리즘만을 고려하여 실제 적용하는 데이터 형이나 기본 함수에 대한 고려가 부족했다.

**4. 실험 환경**

OS: Window 10 Education

CPU: Intel® Core ™ i9-990K CPU @ 3.60GHz 3.60GHz

RAM: 32.0GB

Compiler Visual Studio 2022 Release Mode