

알고리즘설계 HW #1

Sorting

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2020년 5월 1일

학부 정보통신공학과

학년 4

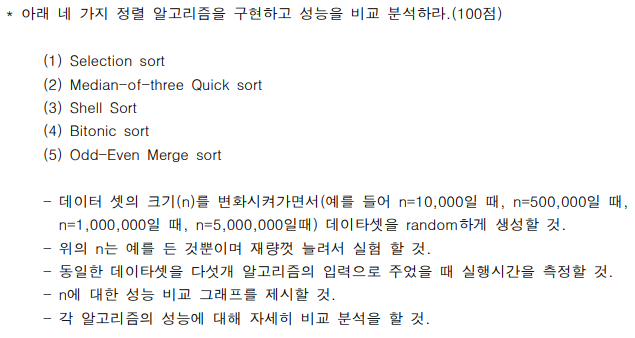
성명 이정우

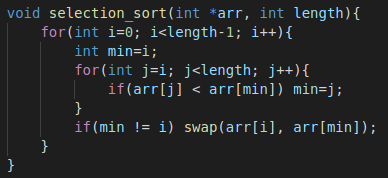
학번 12171833

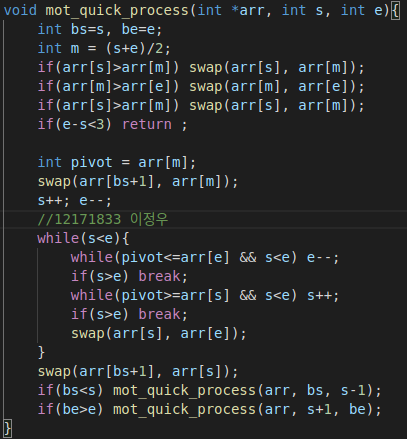
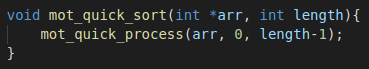


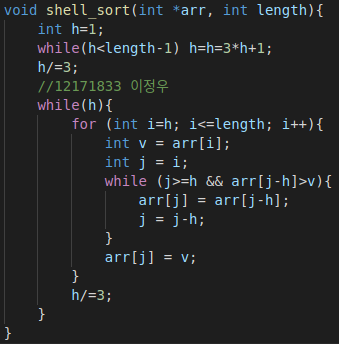
최소한 아래의 내용을 포함할 것.

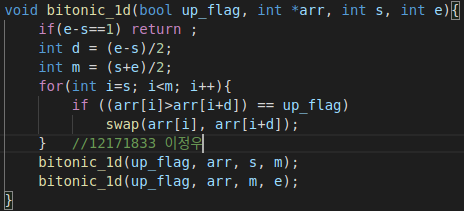
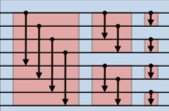
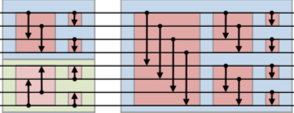
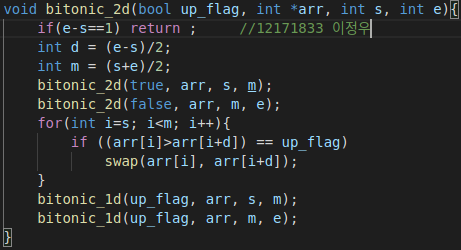
1. 개요



1. 상세 설계내용
   * + - 1. Selection sort  
            

(2) Median-of-three Quick sort  
  
맨 처음 array의 처음, 중간, 끝 값을 비교하여 중간 값을 pivot으로 사용한다.  
처음과 끝 값 사이의 차이가 3보다 작을 경우 pivot을 찾는 과정에서 정렬의 완료되기 때문에 return 해준다.  
중간 값의 위치를 처음+1의 위치로 옮기고 2개의 값은 이미 비교를 하였기 때문에 처음+1, 끝-1부터 pivot과 비교를 하여 값을 swap해준다.  
마지막에 pivot값을 배열의 pivot보다 작은 값들의 맨 끝으로 옮기고 pivot의 좌, 우 배열을 quick sort하는 것을 반복한다.   


(3) Shell sort  
  
맨 처음 h의 크기를 찾은 후 h의 크기가 1의 될 때까지 3으로 나누어 주며  
h의 간격대로 insertion sort를 수행한다.

(4) Bitonic sort  
   
배열의 s~e-1사이의 값을 같은 방향으로 array의 크기를 반으로 나누며 반복 정렬한다.  


4

5

2

3

1

3

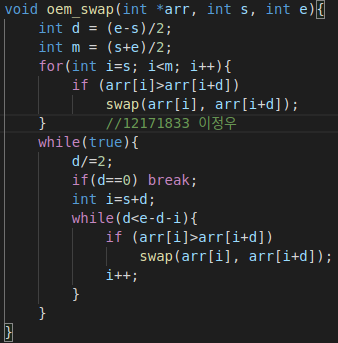
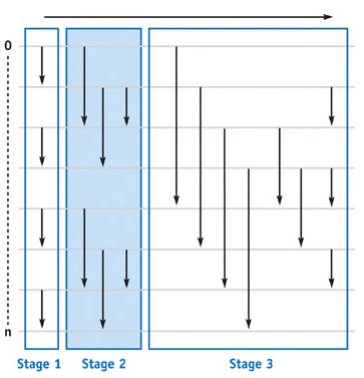
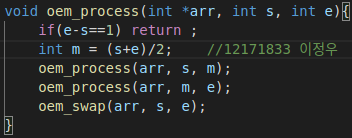
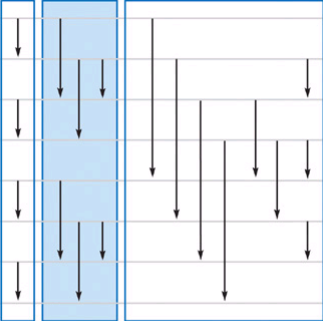
5

4

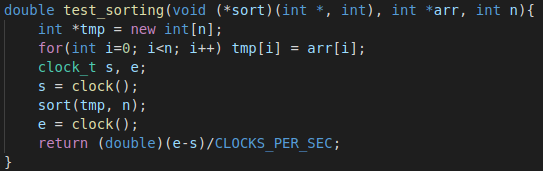
1

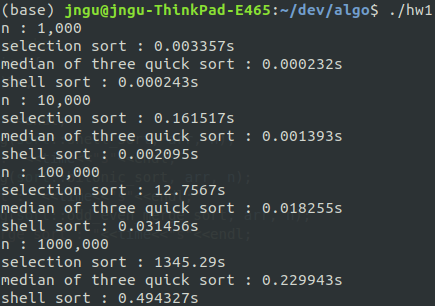
2

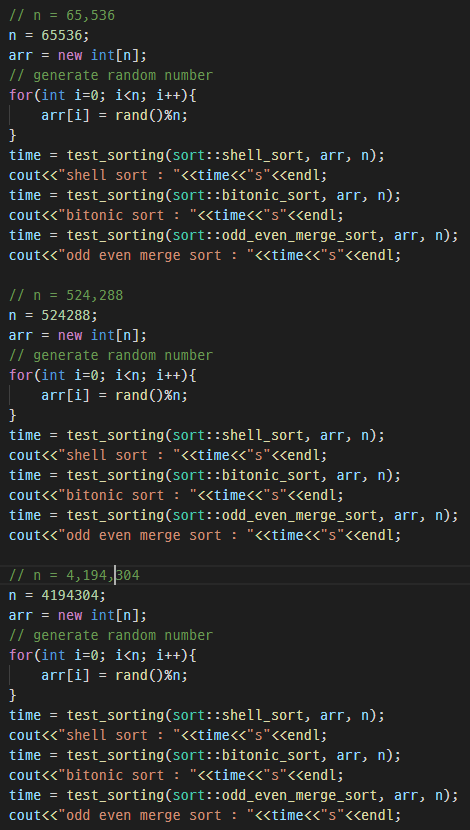
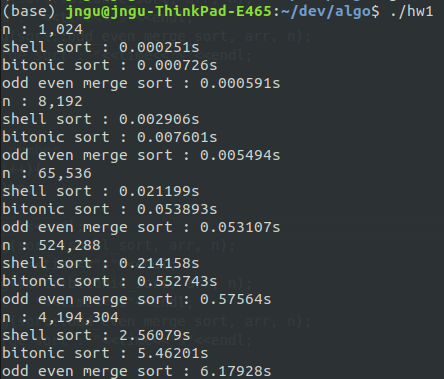
처음의 배열을 각기 다른 방향으로 반 씩 나누어 bitonic\_2d를 불러준다.   
다음으로 자기 자신을 up\_flag방향으로 정렬하고, 그 bitonic\_1d를 불러주어 반으로 나누어가며 up\_flag방향으로 계속 정렬하도록 한다.  


(5) Odd-Even Merge sort  
   
처음에 (e-s)/2의 간격 만큼으로 s부터 e까지 정렬을 한 후  
d를 2로 나눠가면서 s+d부터 s-d까지 d간격으로 정렬을 d가 0보다 큰 동안 반복한다.   
   


1. 실행 화면



(1), (2), (3)번 sorting 비교  
  


(3), (4), (5)번 sorting 비교  
  


1. 분석 및 결론

(1), (2), (3)번 sorting 비교

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sort/Data number | **1,000** | **10,000** | **100,000** | **1,000,000** |
| **Selection sort** | 0.003357 (s) | 0.161517 (s) | 12.759700 (s) | 1345.290000 (s) |
| **Median-of-three Quick sort** | 0.000232 (s) | 0.001393 (s) | 0.018255 (s) | 0.229943 (s) |
| **Shell sort** | 0.000243 (s) | 0.002095 (s) | 0.031456 (s) | 0.494327 (s) |

Selection sort는 O(n2), Median-of-three Quick sort는 O(nlogn), Shell sort는 O(n3/2)의 평균 시간복잡도를 가진다.

데이터 증가에 따른 시간의 변화를 살펴보면

(1) Selection sort : 데이터가 10배 증가할 때 시간이 48배, 78배, 105배 증가한다. 데이터의 양이 많아질수록 시간복잡도와 근사하게 시간이 증가하는 것을 알 수 있다.

(2) Median-of-three Quick sort : 이론상 데이터가 늘어날 때 13.3배, 12.5배, 12배로 늘어나야 하는데 실제 측정 결과 시간이 6배, 13배, 12배늘어나는 것을 확인할 수 있었다.

(3) Shell sort : 이론상 데이터가 늘어날 때 시간이 약 30배 증가하여야 하는데 실제 측정결과 시간이 8.5배, 15배, 15.7배

(3), (4), (5)번 sorting 비교

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sort/Data number | **1,024** | **8,192** | **65,536** | **524,288** | **4,194,304** |
| **Shell sort** | 0.000251 (s) | 0.002906 (s) | 0.021199 (s) | 0.214158 (s) | 2.56079 (s) |
| **Bitonic sort** | 0.000726 (s) | 0.007601 (s) | 0.053893 (s) | 0.552743 (s) | 5.46201 (s) |
| **Odd-Even Merge sort** | 0.000591 (s) | 0.005494 (s) | 0.053017 (s) | 0.575640 (s) | 6.17928 (s) |

Shell sort는 O(n3/2), Bitonic sort는 O(nlog2n), Median-of-three Odd-Even Merge sort는 O(nlog2n)의 평균 시간복잡도를 가진다.

(3) Shell sort : 이론상 데이터가 늘어날 때 시간이 약 22.6배 증가하여야 하는데 실제 측정결과 시간이 11.6배, 7.3배, 10.1배, 12배 증가하였다.

(4) Bitonic sort : 이론상 데이터가 늘어날 때 13.5배, 12.1배, 11.3배, 10.7배로 늘어나야 하는데 실제 측정 결과 시간이 10.5배, 7.1배, 10.3배, 9.9배늘어나는 것을 확인할 수 있었다.

(5) Odd-Even Merge sort : 이론상 데이터가 늘어날 때 13.5배, 12.1배, 11.3배, 10.7배로 늘어나야 하는데 실제 측정 결과 시간이 9.3배, 9.6배, 10.8배, 10.7배늘어나는 것을 확인할 수 있었다.

측정해본 결과 데이터가 적을 때는 Big-Oh보다 작게 나왔고 데이터의 개수가 점점 많아 질수록 평균 시간복잡도에 맞는 것을 볼 수 있었다.

Bitonic sort와 Odd-Even Merge sort의 경우 gpu야 amd제품이기 때문에 cuda코드로 테스트를 해보진 못했지만 그렇지 않을 때의 결과가 이론과 비슷하게 나왔다.

Shell sort의 경우 이론의 값보다 작게 나오긴 하였지만 값에 큰 차이가 있었는데 실제로 nlogn과 더 근사한 경향을 보여주었다. 하지만 nlogn보다는 좀 더 큰 값을 가졌다.