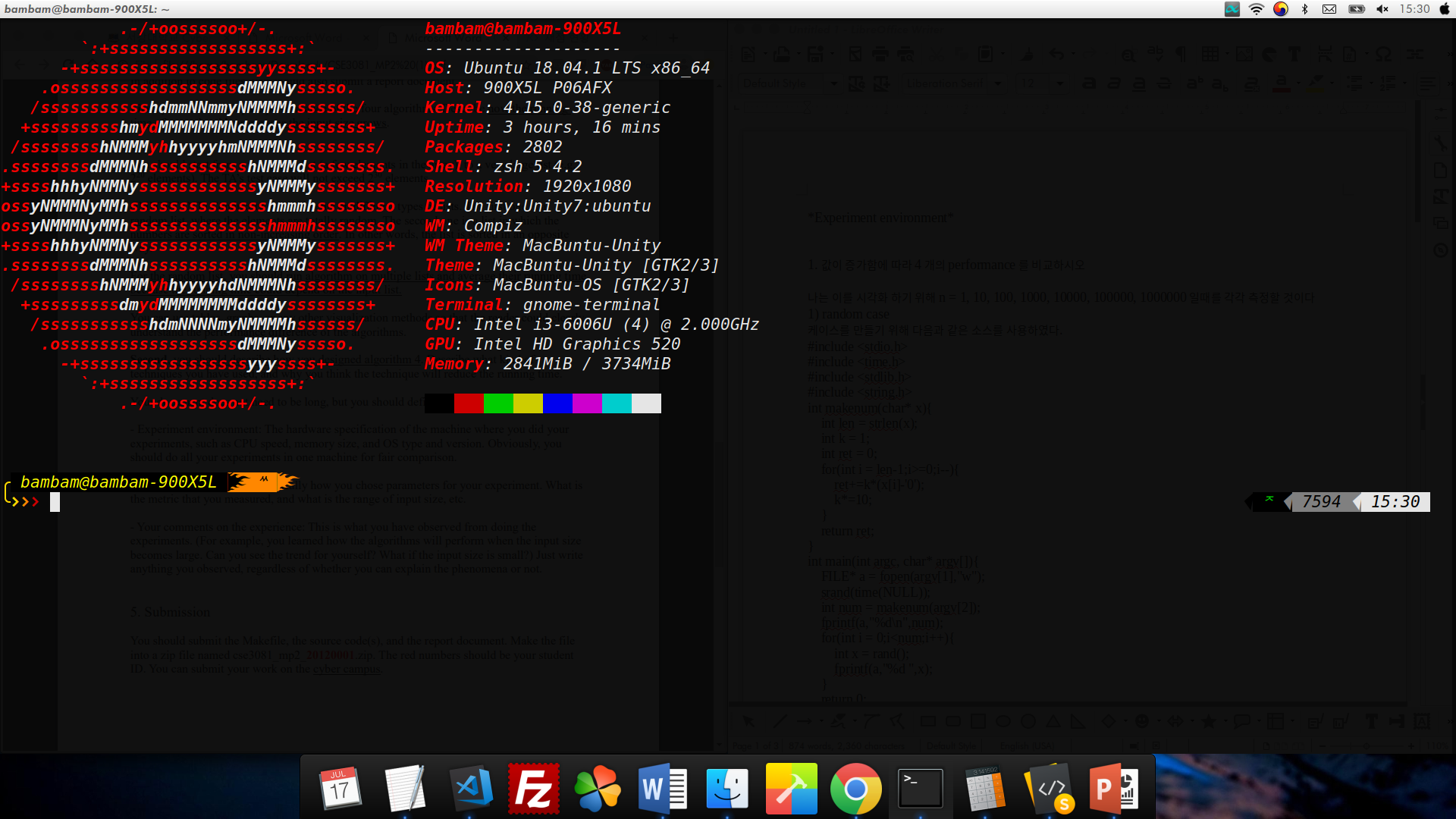
20171687 전민수

\*Experiment environment\*

실행 환경은 위의 그림에 잘 나와있다.

개인 노트북의 ubuntu에서 실험을 진행하였다.

\*Experiment setup\*

binary file name : mss20171687

실행할 떄: ./mss20171687 input.txt 1

첫번째 parameter : input file name

두번쨰 parameter : 알고리즘 분류

1. insertion sort

2. quick sort

3. heap sort

4. mysort(introsort + counting sort)

input size : n<2^30, value < 2^31

측정하고자 하는 것: input size에 대한 정렬 알고리즘의 성능 비교

\*Experiment\*

1. 값이 증가함에 따라 4개의 performance를 비교하시오

나는 이를 시각화 하기 위해 n = 1, 10, 100, 1000, 10000, 100000, 1000000일때를 각각 측정할 것이다

1) random case

케이스를 만들기 위해 다음과 같은 소스를 사용하였다.

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <string.h>

int makenum(char\* x){

int len = strlen(x);

int k = 1;

int ret = 0;

for(int i = len-1;i>=0;i--){

ret+=k\*(x[i]-'0');

k\*=10;

}

return ret;

}

int main(int argc, char\* argv[]){

FILE\* a = fopen(argv[1],"w");

srand(time(NULL));

int num = makenum(argv[2]);

fprintf(a,"%d\n",num);

for(int i = 0;i<num;i++){

int x = rand();

fprintf(a,"%d ",x);

}

return 0;

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 1 | 3 | 48 | 2797 | 312349 | 27814055 | 오래걸림 |
| 2 | 2 | 3 | 28 | 421 | 11069 | 62183 | 640474 |
| 3 | 10 | 23 | 150 | 1738 | 6100 | 63361 | 699835 |
| 4 | 8 | 32 | 35 | 1230 | 4166 | 51893 | 576377 |

시간의 단위는 10^-6초가 기준이다.

2) 내림차순으로 정렬된 case

이 케이스를 만들기 위해 다음과 같은 소스를 사용하였다.

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <string.h>

int makenum(char\* x){

int len = strlen(x);

int k = 1;

int ret = 0;

for(int i = len-1;i>=0;i--){

ret+=k\*(x[i]-'0');

k\*=10;

}

return ret;

}

int main(int argc, char\* argv[]){

FILE\* a = fopen(argv[1],"w");

srand(time(NULL));

int num = makenum(argv[2]);

fprintf(a,"%d\n",num);

int t = 5\*num;

for(int i = 0;i<num;i++){

int x = rand()%5;

t-=x;

fprintf(a,"%d ",t);

}

return 0;

}

이를 토대로 똑같이 표를 그려보면 다음과 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 2 | 9 | 60 | 8215 | 568773 | 55721112 | 오래걸림 |
| 2 | 3 | 4 | 87 | 8556 | 788638 | 79494876 | 오래걸림 |
| 3 | 5 | 26 | 38 | 1214 | 8834 | 47967 | 496016 |
| 4 | 9 | 3 | 10 | 50 | 475 | 4773 | 48463 |
| 4’ | 9 | 58 | 56 | 1092 | 7407 | 39178 | 425749 |

4’은 counting sort를 합치지 않은 알고리즘의 시간이다.

시간의 단위는 10^-6초가 기준이다.

2. 알고리즘 4를 어떻게 디자인하였는지 설명하시오.

ppt에 나와있는 introsort는 ppt에 나와있는 퀵소트의 모든 최적화 기법들을 넣어서 만들어진 소트라고 생각하여, 나는 introsort를 기반으로 알고리즘 4를 디자인 하였다.

다만, 만약 배열에서 나온 값 중 최댓값과 최솟값의 차가 nlogn을 넘지 않는다면, 그 숫자들을 모두 배열에 넣고 뺴는 과정을 통해 O(N) (N : 최댓값-최솟값)만에 정렬을 해낼 수 있기 때문에, 만약 최댓값과 최솟값의 차가 2\*nlogn보다 작다면, counting sort기법을 활용하여 정렬하였다.

나의 introsort는 다음과 같다.

1) 정렬은 32를 기준으로 블럭 단위로 나뉘어 진다. 이후, insertion sort를 통해 최종 정렬을 수행한다.

블럭들은 각각 quick sort의 partition 원리에 의해 작은 블럭부터 큰 블럭까지 정렬된 상태로 있을 것이고, 각각의 블럭을 정렬하는데에는 32가 상수이기 떄문에 상수 시간이 걸리므로 O(n)의 시간이 걸린다.

2) 깊이의 제한을 두고, 그 깊이를 넘어가면 heapsort를 활용하여 정렬을 한다. 이를 통해 정렬을 O(nlogn)의 시간복잡도를 보장받을 수 있다.

3)랜덤하게 s와 e사이에 있는 숫자 중 하나를 선택 한 후, 그것의 중간값을 찾고 그것을 기준으로 partition을 나눈다. 이를 통해 최악의 케이스에서도 괜찮은 pivot값을 얻을 확률이 높아진다.

4) tail recursion을 통해 재귀로 인하여 먹히는 메모리가 줄어들 수 있도록 한다.

\*comments on the experience\*

1) insertion sort는 두번째 테스트 케이스가 무조건 최악의 케이스일 수밖에 없다. 왜냐하면 insertion sort는 어떤 한 점에서 그 점 앞은 모두 정렬되어 있는 상태에서 그 점이 들어갈 위치를 결정하여 넣어주는 알고리즘인데, 앞의 정렬되어 있는 모든 숫자들보다, 현재 점이 더 작기 떄문에, 모든 숫자들을 돌아봐야 하기 때문이다.

따라서 랜덤케이스에 비해 내림차순으로 정렬되어 있는 케이스에서 거의 2배 가량의 시간이 늘어남을 확인 할 수 있었다.

2) quick sort 또한, 아무런 optimization이 없는 상황 속에서는 두번째 케이스가 무조건 최악의 케이스이다. 나의 quick sort는 end값을 기준으로 partition을 하였다. 내림차순으로 되어 있는 리스트에서 s,e에서 e에는 가장 작은 값이 저장되어 있다. 따라서 한번 파티션을 하면 pivot의 왼쪽에는 0개, 오른쪽에는 n-1개가 있을 것이기 때문에, 이를 계속하면 최악의 케이스인 n^2가 나온다. 따라서 랜덤일때에는 nlogn으로 수렴하듯 heapsort와 비슷한 시간을 보였던 반면, 내림차순일때에는 insertion sort보다도 시간이 더 오래걸리는 것을 관찰 할 수 있었다.

3) heap sort는 min heap을 구성한 후, 그것을 하나하나 뽑으면서 정렬을 하는 구조이다. Min heap을 구성하는데 O(n), 그것을 하나하나 뽑는데 O(nlogn)이 걸려 총 O(nlogn)이 보장되는 소트이다. 다만, 힙을 구성할 새로운 배열 하나를 선언하여 메모리를 좀 더 먹었다.

4) 알고리즘의 디자인은 위에서 다 설명하였다.

실험 결과를 보면, n이 작을때에는 다른 소트보다 오히려 많이 걸리는 모습을 볼 수 있었는데, 이는 rand을 하여 최적에 가까울 확률이 높은 pivot을 뽑기 위해 여러 연산들을 하기 때문이다. 숫자가 커지면서, 그런 작은 연산들이 상관 없어지자, 다른 3개의 알고리즘보다 더 빠른 모습을 보였다.

또한, 랜덤으로 pivoting을 하였기 때문에, 내림차순의 배열에서도 큰 시간의 변화 없이 돌아감을 확인 할 수 있었다.