Data Structure

Assignment #9

povis ID: ljs9904ljs

학번: 20180551

학과: 무은재학부

이름: 이준석

Problem 1 (R-11.11)

최악의 경우일 때 quick sort는 big-Omega (n^2)가 된다. pivot으로 선택되는 것이 매번 가장 큰 값이거나 가장 작은 값일 경우에 quick sort는 최악의 경우를 겪는다. 왜냐하면 quick sort는 pivot을 기준으로 작은 것들을 왼쪽 리스트에, 큰 것들을 오른쪽 리스트에 나누어 놓고 정렬하기에 효율적인 알고리즘인데 매번 가장 큰 것이나 가장 작은 것을 고른다면 원래의 리스트가 분할되지 않기 때문이다.

n/2의 floor를 pivot으로 선택하는 quick sort이므로 매 경우마다 골라지는 그 n/2의 floor의 값이 그 리스트에서 최대의 값인 경우를 생각해보자.

{ 3, 5, 7, 8, 6, 4, 2, 1 }을 { 1, 2, 3, 4, 5, 6, 7, 8 }로 정렬하고자 할 때가 바로 그런 경우이다.

{357 8 6421} -> 8이 pivot이 되고

{35 7 6421} -> 7이 pivot이 되고

{35 6 421} -> 6이 pivot이 되고

{3 5 421} -> 5가 pivot이 되고

{3 4 2 1} -> 4가 pivot이 되고

{3 2 1} -> 3이 pivot이 되고

{2 1} -> 2가 pivot이 되고

{1}

합치면

1 2 3 4 5 6 7 8

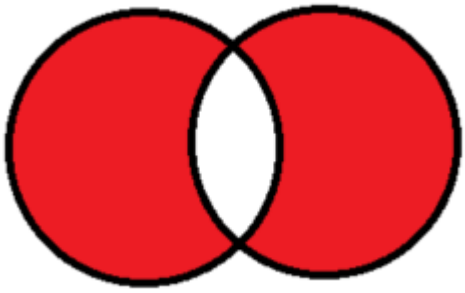
Problem 2 (R-11.22)

merge sort와 heap sort는 언제나 시간 복잡도가 O(nLogn) 이다. 하지만 insertion sort는 최선의 경우에 O(n)이고 최악의 경우에 O(n^2)이다. insertion sort에서 최선의 경우는 정렬하고자 하는 대상이 이미 정렬이 되어 있는 경우일 때이다.

즉, 우리가 왼쪽에서 오른쪽으로 오름차순 정렬을 하고자 할 때 주어진 자료가 {1,2,3,4,5} 라면 insertion sort는 O(n)이고 merge sort와 heap sort는 O(nLogn) 이다. 만약 이 자료가 정반대로 뒤집어진다면, {5,4,3,2,1}이 될 것이고 정렬되지 않은 자료가 되어버린다. 따라서 insertion sort는 O(n^2)이고 merge sort와 heap sort는 O(nLogn)이다.

Problem 3 (C-11.3)

문제에서 구하라는 집합을 벤 다이어그램을 통해 나타내면 다음과 같다.



그림에서 색칠한 부분이 문제에서 구하라는 것을 나타낸다.

1. set A와 set B의 union을 구한다. (A . union (B)) 이것을 C라 하자.

2. set A와 set B의 intersect를 구한다. (A . intersect (B)) 이것을 D라 하자.

3. C와 D의 substract를 구한다. C에서 D를 뺀다. (C . substract (D))

Problem 4 (C-11.10)

C-11.10 Describe a nonrecursive, in-place version of the quick-sort algorithm. The

algorithm should still be based on the same divide-and-conquer approach,

but use an explicit stack to process subproblems.

```
void quick_sort(int input[], int start, int end)
{
	int pivot;
	if (start < end) {

		int stack[end - start + 1];
		int top = -1;

		stack[++top] = start;
		stack[++top] = end;

		while (top >= 0) {

			start = stack[top--];
			end = stack[top--];
			int p = partition(arr, start, end);
			if (p - 1 > start) {
				stack[++top] = start;
				stack[++top] = p - 1;
			}
			if (p + 1 < end) {
				stack[++top] = p + 1;
				stack[++top] = end;
			}
		}
	}
	return;
}


int partition(int input[], int start, int end)
{
	int pivot, temp, low, high;

	low = start;
	high = end + 1;
	pivot = input[start];
```

```
	do {
		do {
			low++;
		} while (low <= end && input[low] < pivot);

		do {
			high--;
		} while (high >= start && input[high] > pivot);

		if (low < high) {
			mySwap(&input[low], &input[high]);
		}

	} while (low < high);

	mySwap(&input[start], &input[high]);

	return high;
}


void mySwap(int *a, int *b)
{
	int temp;
	temp = *a;
	*a = *b;
	*b = temp;
}
```

quick_sort가 문제에서 요구하는 정렬을 수행하는 함수이다.

partition은 pivot에 해당하는 값을 정해주기 위한 함수이다.

mySwap은 매개변수로 주어지는 2개의 값을 서로 바꿔주는 함수이다.

Problem 5 (C-11.12)

간단한 in-place sort로는 bubble sort가 있다. 정렬 결과 모든 0이 모든 1 앞에 나오도록 하기 위해서는 bubble sort를 통해 오름차순으로 정렬하면 된다.

```
for( int i=n-1; i>0; i--)
{
   for( int j=0; j<I; j++)
   {
   if(A[j] > A[j+1])
   std::swap( A[j], A[j+1] );
   }
}
```