데이터 구조 Assn2

20180551 이준석

POVIS ID : ljs9904ljs

Problem 1 (R-4.13)

big-oh notation을 이용한다.

시간복잡도가 큰 것을 우선으로 나열하였다.

①$2^n$

②n^3

③$n^2 + 10n$

④$4n\log n + 2n$

⑤$n\log n$

⑥$4n$

⑦$3n + 100\log n$

⑧$2^{\log n}$

⑨$2^{10}$

일반적으로,
[지수 함수 > n^3 > n^2 > nlogn > n > logn > 상수 함수]
인 것을 이용하였다.

Problem 2 (R-4.32)

big-omega의 정의는 다음과 같다.

$f(n)$ is W($g(n)$), that is, there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq cg(n)$, for $n \geq n_0$.

n^2 is big-Omega nlogn을 보이는 것이 문제이다. 따라서 n^2 ≥ c * nlogn 을 만족하는 임의의 c와 n_0을 찾으면 된다.

c = 1 이라 가정하자.

n은 1 이상의 값이므로 1을 대입하면 좌변은 1, 우변은 0이다.
n = 2일 때, 좌변은 4, 우변은 2이다. 순간 변화율을 고려할 때 정수 n에 대하여 n이 logn보다 항상 크므로 n^2이 nlogn보다 더욱 가파르게 증가할 것이다.
따라서 n^2 ≥ 1 * nlogn 은 n≥1에서 항상 성립한다. 즉 가정에 대한 모순이 없으므로 n^2 is big-Omega nlogn

Problem 3 (C-4.2)

```
int isUnique(int arr[],int n) {

	if (n == 1) return true; // 1번

	for (int i = 0; i < n - 1; i++) { // i 초기화 1번, 조건 확인 n번, i 값 변화 2(n-1)
		if (arr[i] == arr[n - 1]) return false; // 3(n-1)번
	}

	return isUnique(arr, n – 1); //1번

} // 한 번의 호출에서 6n – 2 번 실행
```

최악의 경우 n번 호출

$\sum_{k=2}^{n}(6k-2)$ = (6n(n+1) / 2) - 2n –4 = 3n^2 + 3n – 2n – 4 = 3n^2 + n – 4
k = 1 일 때는 1번 이므로

총 3n^2 + n – 3 번 실행

f(n) = 3n^2 + n – 3 이라 하자.
f(n) ≤ c * n^2을 만족하는 c를 찾자.
c = 4 일 때, n ≥ 1에 대해 항상 만족한다.
따라서 최악의 경우에 대해

f(n)은 big-Oh n^2 (n ≥ 1) 이다.

Problem 4 (C-4.17)

Proposition 4.20에 의하여, fibonacci function $F(n) < 2^n$

$f(1) = 1$, $f(2) = 2$, $f(3) = 3$, $f(4) = 5$, $f(5) = 8$, $f(6) = 13$, $f(7) = 21$, $f(8) = 34$, $f(9) = 55$, $f(10) = 89$, $f(11) = 144$

$n = 10$ 일 때, $f(10) = 89 > (3/2)^{10} = 57.6650...$
$n = 11$ 일 때, $f(11) = 144 > (3/2)^{11} = 86.4975...$

$n = k$, $n = k+1$ 일 때 참이라 가정하자.
$f(k) \geq (3/2)^k$, $f(k+1) \geq (3/2)^{(k+1)}$

$n = k + 2$ 일 때는 $f(k+2) \geq (3/2)^{(k+2)}$
$f(k+2) = f(k+1) + f(k) \geq (3/2)^{(k+1)} + (3/2)^k = (5/2)^k > (9/4)^k$
이므로 $n = k + 2$ 일 때 참이다.

따라서 $c = 1$이라 하면
$F(n) = F(n-1) + F(n-2) \geq 1 * (3/2)^{(n)}$ 이 성립하여

$F(n)$은 big-Omega $(3/2)^n$ $(n \geq 10)$ 이다.

Problem 5 (C-4.24)

```
#include <iostream>
#include <cstdio>

using namespace std;

void printArr(int arr[], int n) {
	for (int i = 0; i < n; i++) {
	printf("%d ", arr[i]);
	}
	printf("₩n₩n");
}

int main() {
	int i;
	int n;
	int mode = 0;
	int max = 0;

	scanf("%d", &n); //숫자가 총 몇 개인지를 입력 받는다.

	int* data = (int*)calloc(n, sizeof(int)); //숫자를 저장할 배열
	int* count = (int*)calloc(4*n + 1, sizeof(int)); //몇 번 나타났는지를 저장할 배열

	for (i = 0; i < n; i++) { // data 배열에 숫자를 입력받는다.
		scanf("%d", &data[i]);
	}

	for (i = 0; i < n; i++) { // 몇 번 나타났는지 count한다.
		count[data[i]]++;
	}

	for (i = 0; i < 4*n+1; i++) {
		if (max < count[i]) {
			max = count[i];
			mode = i;
		}
	}

	printf("최빈값은 %d이다. 총 %d 번 출현하였다.₩n", mode, max);

	free(data);
	free(count);

	return 0;
}
```

(위의 코드는 가장 많이 나타나는 값이 두 가지 이상이 아니라 한 가지일 때를 가정한 코드이다.)

(만일 두 가지 이상의 최빈값이 존재한다면 그것을 검사하기 위해 두 번째 for문 안에 조건을 추가해주면 가능하기 때문에 시간 복잡도의 big-Oh notation의 차이는 발생하지 않는다.)

값의 범위가 정해져 있으므로 data 배열을 count 배열의 index로 보았다. 그렇게 하는 것으로 인해서 이중 for문이 아닌, 단일 for문 2개로 문제를 해결하는 것이 가능해졌다. 단일 for문 2개이므로 O(n)의 시간 복잡도로 문제해결이 가능하다.