ESAME 18/01/21

3) • MyAlgorithm 1

$T(n) = 4T\left(\frac{n}{2}\right) + n$

$a = 4 \quad b = 2 \quad f(n) = n \qquad d = \log_2 4 = 2 \Rightarrow g(n) = n^2$

CASO 1 DEL TEOREMA MASTER

$f(n) = O(n^{d-\varepsilon}) \quad \exists \varepsilon > 0$

$n = n^{2-\varepsilon} \Rightarrow \varepsilon = 1$

$T(n) = \Theta(n^2)$

• MyAlgorithm 2

$T(n) = 3T\left(\frac{n}{2}\right) + n^2$

$a = 3 \quad b = 2 \quad f(n) = n^2 \qquad d = \log_2 3 < 2 \Rightarrow g(n) = n^{\log_2 3}$

$f(n) > g(n)$

CASO 3 DEL TEOREMA MASTER

$f(n) = \Omega(n^{d+\varepsilon}) \quad \exists \varepsilon > 0$

$n^2 = n^{\log_2 3 + \varepsilon} \rightarrow \varepsilon = 2 - \log_2 3 > 0$

$\exists c < 1$ è per n suff. grande a $f\left(\frac{n}{b}\right) \leq c f(n)$

$3\left(\frac{n}{2}\right)^2 \leq c n^2 \rightarrow \frac{3}{4} \cancel{n^2} \leq c \cancel{n^2} \rightarrow \frac{3}{4} \leq c \rightarrow \frac{3}{4} \leq c < 1$

$T(n) = \Theta(n^2)$

Entrambi gli algoritmi hanno complessità asintotica $\Theta(n^2)$, ma il costo non ricorsivo di MyAlgorithm1 è lineare ($\Theta(n)$) e non quadratico ($\Theta(n^2)$).

4)
```
struct Edge {
    int target;
    double weight;
};

bool can_reach_destination(const vector<vector<Edge>> & graph, double C, int s, int d) {
    //Inizializzazione
    unordered_map<int, double> fuel;
    for(size_t i=0; i<graph.size(); ++i){
        fuel[i]=-1.0; //-1 significa non raggiunto
    }
    fuel[s]=C;

    // Max-heap usando min-heap con valori negativi
    priority_queue<pair<double,int>> heap;
    heap.push({fuel[s],s});
    while(!heap.empty()){
        auto current=heap.top();
        heap.pop();
        double current_fuel=current.first;
        int u= current.second;
        if(u==d){
            return true;
        }
        if(current_fuel>=fuel[u]){
            for(const Edge& edge: graph[u]){
                int v=edge.target;
                double w=edge.weight;
                if(current_fuel>=w){
                    double remaining_fuel=current_fuel-w;
                    if(remaining_fuel>fuel[v]){
                        fuel[v]=remaining_fuel;
                        heap.push({remaining_fuel,v});
                    }
                }
            }
        }
    }
    return false;
}
```

ESAME 18/01/21

1)
```
typedef struct node {
    int key;
    node* left;
    node* right;
}*Node;

int k-limitato(Node u, int k) {
    int sum;
    return k-limitato aux(u, K, sum);
}
int k-limitato aux(Node u, int k, int & current_sum) {
    if (u == nullptr) {
        current_sum = 0;
        return 1;
    }
    int left_sum, right_sum;
    int sx = k-limitato aux(u->left, k, left_sum);
    int dx = k-limitato aux(u->right, k, right_sum);
    // Calcola la somma massima dei cammini da questo nodo a una foglia
    current_sum = u->key + (left_sum > right_sum ? left_sum : rightsum);

    //Verifica se entrambi i sottoalberi sono k-limitati e se
    return sx && dx && (current_sum ≤ k);
}
```

COMPLESSITÀ

$T(n) = O(n)$

RELAZIONE DI RICORRENZA

La relazione di ricorrenza per il tempo di esecuzione è: $T(n) = T(m) + T(n-m-1) + O(1)$, dove m è il numero di nodi nel sottoalbero sinistro.

Questa è la stessa relazione degli algoritmi di visita degli alberi, che si risolve in $O(n)$ poiché ogni nodo viene visitato una sola volta.

2)
```
int scarto (int n, int punteggi) {
    // Ordina l'array utilizzando il mergesort
    mergeSort (punteggi, 0, n-1);
    int totale_scarto = 0;
    for (int=0; i<n; i+=2){
        totale_scarto += punteggi[i+1] - punteggi[i];
    }
    return totale scarto;
}
```

$T(n) = \underbrace{O(n \log n)}_{\text{MERGESORT}} + \underbrace{O(n)}_{\substack{\text{CALCOLO} \\ \text{SCARTI}}} = O(n \log n)$