Test 3 Parcial - Manuel Morales

Q1. If I know my problem has subproblems, which possible algorithm techniques can I use?

We can use the following techniques:

- Greedy Algorithm
- Divide and Conquer
- Dynamic Programming And maybe more techniques, but I only know these three.

Q2. Given T = xyzasxyzxmb and P = xyzxm. Define the alphabet, finite set of states, start date, and accepting state.

The alphabet is Σ = {x, y, z, a, s, m, b} The finite set of states is Q = {q0, q1, q2, q3, q4, q5, q6, q7} The start state is q0 The accepting state is q5

Q3. Given T = xyzasxyzxmb and P = xyzxm. What is the value of the transition function S(2, x)

The value of the transition function S(2, x) is 3

Q4. Cube is in a food competition; he is given N meals and has to eat as much as possible in M minutes. Cube can only eat one meal at a time and knows how much time he needs to finish eating each meal. Find the maximum number of meals he can eat in M minutes.

Input: A number N and M **Output:** The maximum number of meals he can eat.

Sample Input: 5 10 6 3 2 4 1 Sample Output: 4

a. Identify if this problem can be solved with dynamic programming and/or greedy algorithm.

a. If you say that can be solved with dynamic programming.

- i. Analyze the problem based on subproblems. This problem can be broken down into subproblems by considering the weight of the knapsack. Each subproblem represents the maximum number of meals that can be eaten in M minutes. The subproblems can be solved by considering the time needed to eat each meal and the time available to eat the meals.

- ii. Identify where the overlap occurs The overlap occurs when we are calculating the maximum number of meals that can be eaten in M minutes. The maximum number of meals that can be eaten in M minutes is the maximum of the maximum number of meals that can be eaten in M-1 minutes and the maximum number of meals that can be eaten in M-2, M-3, ..., 1 minutes.

b. If you say that can be solved with greedy algorithm.

- i. Identify the greedy choice. The greedy choice in this problem is to eat the meals that take the least amount of time to eat first. By eating the meals that take the least amount of time to eat first, Cube can eat more meals in M minutes.
- ii. Explican the optimal substructure. The optimal substructure is satisfied because if we can find the optimal way to eat a meal in a L1 based on the time needed to eat each meal, the global optimal solution will contain the optimal solution for a (G - L1) and the optimal solution of a L1

b. If you say that problem can not be solved with a greedy algorithm or dynamic programming, explain the reason using and example.

The problem can be solved with both algorithms

c. If you say the problem can be solved with dynamic programming, implement your solution using dynamic programming; otherwise, implement your solution with a greedy algorithm.

Greedy Solution

```
#ifdef LOCAL
#include "/home/morven/Desktop/code/competitive-programming/conf/debug.h"
#define line cerr << "--------------------" << endl;
#else
#define deb(x...)
#define line
#endif

#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
#define F first
#define S second
#define pb push_back
#define sz size
#define all(x) begin(x), end(x)
#define sortt(x) sort(all(x))
#define each(x, xs) for (auto &x : (xs))
#define rep(i, be, en) for (__typeof(en) i = (be); i < (en); i++)
#define per(i, be) for (__typeof(be) i = (be)-1; i >= 0; i--)
#define readline(x) getline(cin, x)
#define strInt(str) stoi(str)
#define chrInt(chr) (int)chr - 48
#define ensp(i, n) (" \n"[i == n - 1])

template <typename... T>
void cinn(T &...args)
{
  ((cin >> args), ...);
}
```

```
template <typename... T>
void coutt(const T &...args)
{
  __typeof(sizeof...(T)) i = 1;
  ((cout << args << (i++ != sizeof...(T) ? " " : "")), ...);
  cout << '\n';
}

template <typename T>
void couts(const T &xs)
{
  for (__typeof(xs.sz()) i = 0; i < xs.sz(); i++)
  {
    cout << xs[i] << " \n"[i == xs.sz() - 1];
  }
}

using ll = long long;
using ld = long double;
using lli = long long int;
using pi = pair<int, int>;
using pl = pair<ll, ll>;
using ti = tuple<int, int, int>;
using tl = tuple<ll, ll, ll>;
using vi = vector<int>;
using vb = vector<bool>;
using vl = vector<ll>;
using vs = vector<string>;
using vvi = vector<vector<int>>;
using vvl = vector<vector<ll>>;
using vpi = vector<pair<int, int>>;
using vpl = vector<pair<ll, ll>>;
template <class T>
using pq = priority_queue<T>;
template <class T>
using pqg = priority_queue<T, vector<T>, greater<T>>;

const ll INF = INT64_MAX;
const int inf = INT32_MAX;
const ld PI = acos(-1);
const lli MOD = 1e9 + 7;
const vector<int> DX{1, 0, -1, 0}, DY{0, 1, 0, -1};
ll testId = 0;

void _()
{
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
#ifdef LOCAL
  freopen("/home/morven/Desktop/code/competitive-programming/conf/main.in",
"r", stdin);
  freopen("/home/morven/Desktop/code/competitive-
programming/conf/main.out", "w", stdout);
```

```
  freopen("/home/morven/Desktop/code/competitive-
programming/conf/main.err", "w", stderr);
#endif
}

void solve();

void init();

void exit();

int main()
{
  _();
  init();
  ll T = 1;
  // cinn(T);
  rep(t, 0, T)
  {
    testId++;
    solve();
  }
  return 0;
}

const ll MAXN = 202020;

void init()
{
}

void exit()
{
}

void solve()
{
  int n, m;
  cinn(n, m);
  vi a(n);
  rep(i, 0, n)
  {
    cinn(a[i]);
  }

  sortt(a);
  int count = 0, index = 0;
  while (index < n)
  {
    if (m >= a[index])
    {
      m -= a[index++];
      count++;
    }
    else
    {
      break;
    }
  }
  coutt(count);
}
```

Dynamic Programming Solution

```
#ifdef LOCAL
#include "/home/morven/Desktop/code/competitive-programming/conf/debug.h"
#define line cerr << "--------------------" << endl;
#else
#define deb(x...)
#define line
#endif

#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
#define F first
#define S second
#define pb push_back
#define sz size
#define all(x) begin(x), end(x)
#define sortt(x) sort(all(x))
#define each(x, xs) for (auto &x : (xs))
#define rep(i, be, en) for (__typeof(en) i = (be); i < (en); i++)
#define per(i, be) for (__typeof(be) i = (be)-1; i >= 0; i--)
#define readline(x) getline(cin, x)
#define strInt(str) stoi(str)
#define chrInt(chr) (int)chr - 48
#define ensp(i, n) (" \n"[i == n - 1])

template <typename... T>
void cinn(T &...args)
{
  ((cin >> args), ...);
}

template <typename... T>
void coutt(const T &...args)
{
  __typeof(sizeof...(T)) i = 1;
  ((cout << args << (i++ != sizeof...(T) ? " " : "")), ...);
  cout << '\n';
}

template <typename T>
void couts(const T &xs)
```

```
{
  for (__typeof(xs.sz()) i = 0; i < xs.sz(); i++)
  {
    cout << xs[i] << " \n"[i == xs.sz() - 1];
  }
}

using ll = long long;
using ld = long double;
using lli = long long int;
using pi = pair<int, int>;
using pl = pair<ll, ll>;
using ti = tuple<int, int, int>;
using tl = tuple<ll, ll, ll>;
using vi = vector<int>;
using vb = vector<bool>;
using vl = vector<ll>;
using vs = vector<string>;
using vvi = vector<vector<int>>;
using vvl = vector<vector<ll>>;
using vpi = vector<pair<int, int>>;
using vpl = vector<pair<ll, ll>>;
template <class T>
using pq = priority_queue<T>;
template <class T>
using pqg = priority_queue<T, vector<T>, greater<T>>;

const ll INF = INT64_MAX;
const int inf = INT32_MAX;
const ld PI = acos(-1);
const lli MOD = 1e9 + 7;
const vector<int> DX{1, 0, -1, 0}, DY{0, 1, 0, -1};
ll testId = 0;

void _()
{
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
#ifdef LOCAL
  freopen("/home/morven/Desktop/code/competitive-programming/conf/main.in",
"r", stdin);
  freopen("/home/morven/Desktop/code/competitive-
programming/conf/main.out", "w", stdout);
  freopen("/home/morven/Desktop/code/competitive-
programming/conf/main.err", "w", stderr);
#endif
}

void solve();

void init();

void exit();
```

```
int main()
{
  _();
  init();
  ll T = 1;
  // cinn(T);
  rep(t, 0, T)
  {
    testId++;
    solve();
  }
  return 0;
}

const ll MAXN = 202020;

void init()
{
}

void exit()
{
}

int knapsack(int n, int w, vi &values, vi &weights)
{
  vvi dp(n + 1, vi(w + 1, 0));
  for (int i = 1; i <= n; i++)
  {
    for (int j = 1; j <= w; j++)
    {
      if (weights[i - 1] <= j)
        dp[i][j] = max(dp[i - 1][j], 1 + dp[i - 1][j - weights[i - 1]]);
      else
        dp[i][j] = dp[i - 1][j];
    }
  }
  return dp[n][w];
}

void solve()
{
  int n, m;
  cinn(n, m);
  vi values(n, 0);
  vi weights(m, 0);
  rep(i, 0, n)
  {
    cinn(values[i]);
  }

  rep(i, 0, m)
  {
    weights[i] = i + 1;
  }

  deb(values, weights);
  coutt(knapsack(n, m, values, weights));
}

```

d. What is the time complexity of your solution?

Time complexity Greedy Algorithm: O(nlogn)

Time complexity Dynamic Programming: O(n*m)