Chocolate Bar Greedy Analysis - Manuel Morales

Greedy Choice:

- The greedy choice in this problem is to get as many pieces of different sizes as possible from the chocolate bar. To maximize the number of pieces, we should cut the chocolate bar into pieces of increasing size, initially cutting the chocolate bar into pieces of size 1, then 2, then 3, and so on until the sum of these parts equals or exceeds this size of the chocolate.

Optimal Substructure:

- The optimal substructure is satisfied because if we can find the optimal way to cut the chocolate bar to a L1 based on the size of the chocolate bar, the global optimal solution will contain the optimal solution for a (G - L1) and the optimal solution of an L1.

Implementation:

```
#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 greedy(int &N)
{
  int count = 0;
  int i = 1;
  while (N > 0)
  {
    if (N < i)
      break;
    N -= i;
    i++;
    count++;
  }
  return count;
}

void solve()
{
  int n;
  cinn(n);
```

```
  coutt(greedy(n));
}
```

Time Complexity:

The time complexity of this solution is O(n) because the greedy algorithm iterates through the chocolate bar at most n times, where n is the length of the chocolate bar. The greedy algorithm is efficient because it only requires a single pass through the chocolate bar to determine the maximum number of pieces that can be cut.