10100011

$2^2 \geq 7+2+1 = 9$

$2^4 = 16 \geq 7+4+1 = 12$

$2^0 = P1$

$2^1 = P2$

$2^2 = P4$

$2^3 = P8$

$2^4 = P16$

Set-01

1

Item A, ratio $= \frac{30}{5} = 6$

Item B, ratio $= \frac{50}{10} = 5$

Item C, ratio $= \frac{60}{15} = 4$

Item D, ratio $= \frac{70}{7} = 10$

Item, E, ratio $= \frac{40}{8} = 5$

Ranking item based on ration (Descending order)

Item	D	A	B	E	C
Weight (kg)	7	5	10	8	15
Value($)	70	30	50	40	60
ratio	10	6	5	5	4

2 and (n): 20 kg capacity:

Greedy picks in ratio order

~~D full (7 kg, value)~~

Take D full: weight 7 kg, value 70$, remaining capacity 13 kg

Take A full:

weight 5 kg, value 30$, remaining capacity 8 kg

Take ~~B full~~, frac:

weight 8 kg, value 40$, remaining cap 0 kg

Total for 20 kg:

From D: 7 kg, 70$

From A: 5 kg, 30$

From E: 8 kg, 40$

Maximum value = 70 + 30 + 40 = 140

(4)

```
#include <bits/stdc++.h>

using namespace std;

struct Item{
    char name;
    double weight, value, ratio;
};

int main(){
    vector<Item> items = {
        {'A', 5, 30},
        {'B', 10, 50},
        {'C', 15, 60},
        {'D', 7, 70},
        {'E', 8, 40}
    }

double capacity = 20;

for(auto &itm: items){
    itm.ratio = itm.value/itm.weight;
}
```

```
sort( items.begin(), items.end(), [](const Item
                                     Knapsack
      &a, const Item &b){
          return a.ration > b.ratio;
      });

double remaining = capacity.
double totalValue = 0.0;

cout << fixed << setprecision(2);
cout << "Capacity :" << capacity << " kg\n";
cout << "Take items (possibly fractional) : \n";

for(auto &itm : items){
    if(remaining <= 0)
        break;
    if( itm.weight <= remaining){
        cout << "Take full item " << itm.name
             << " (w=" << itm.weight << ", v=" << itm.
             value << ")\n";
        remaining -= itm.weight;
        totalValue += itm.value;
    }else{
        double frac = remaining / itm.weight;
```

```
        double w = remaining;
        double v = frac * itm.value;
        cout << "Take " << frac * 100 << "% of item "
             << itm.name;
        cout << " (w=" << w << ", v=" << v << ")\n";
        totalValue += v;
        remaining = 0;
    }
}
cout << "Maximum total value = " << totalValue << endl;
return 0;
}
```

Set - 02

(1) Activities : A(1,4), B(3,5), C(0,6), D(5,7)
E(8,9); F(5,9), G(6,10)

Greedy Approach:

→ Pick (A1,4) first (earliest finish)
→ Next that starts ≥ 4. But B(3,5) starts 3<4, so skip B.

Sort by finish time:

Activity	Start Time	Finish time
A	1	4
B		

→ ~~Pick~~ D(5,7) starts $5 \geq 4$ → Pick D.

→ E(8,9) starts $8 \geq 7$ → pick E

→ F(5,9) starts $5 \not\geq 9$ → skip F

→ G(6,10) starts $6 \not\geq 9$ → skip G

∴ Selected Activities : A, D, E (3 activities)

Activity - A: first (earliest finish 4) → Pick A

Activity - B: starts $3 \not\geq$ finish 4 → skip B

Activity - C: starts $0 \not\geq$ finish 4 → skip C

Activity - D: starts $5 \geq 4$ finish 4 → pick D
updated finish 7

Activity - E: starts $8 \geq$ finish 7 → Pick E
updated finish 9

Activity - F: starts $5 \not\geq$ finish 9 → skip F

Activity - G: starts $6 \not\geq$ finish 9 → skip G

picked Activity A, D, E

(2) same as ans-1

(3)
```
#include <bits/stdc++.h>
using namespace std;

struct Activity{
    char name;
    int start, finish;
};
int main(){
  vector<Activity> acts = {
        {'A', 1, 4},
        {'B', 3, 5},
        {'C', 0, 6},
        {'D', 5, 7},
        {'E', 8, 9},
        {'F', 5, 9},
        {'G', 6, 10}
  };
```

```
sort (acts.begin(), acts.end(), [](const Activity &a,
        const Activity &b) {
            return a.finish < b.finish;
    });

cout << "Selected Activities : \n";
int lastfinish = -1;
for(auto &a : acts) {
    if (a.start >= lastfinish) {
        cout << a.name << " (" << a.start << ","
        << a.finish << ")\n";
        lastfinish = a.finish;
    }
}
return 0;
}
```

Set-03

(1.)

(2)

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

int lcsLength(const string &a, const string &b){
    int m = a.size(), n = b.size();

    vector<vector<int>> dp(n+1, vector<int>(m+1,0));

    for(int i = 1; i <= m; i++){
        for(int j=1; j<=n; j++){
            if(a[i-1] == b[j-1]){
                dp[i][j] = dp[i-1][j-1] + 1;
            }
            else{
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }
}
return dp[m][n]

}

int main(){

}
```

(3)

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

string lcsString(const string &a, const string &b){
    int m = a.size(), n = b.size();
    vector<vector<int>> dp(n+1, vector<int>
                           (m+1, 0));

    for(int i=1; i<=m; i++){
        for(int j=1; j<=n; j++){
            if(dp[i][j] == dp
            if(dp[i-1][j-1] ==
            if( a[i-1] == b[j-1]){
                dp[i][j] = dp[i-1][j-1]+1;
            }else{
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]
            }
        }
    }

    string lcs = "";
    int i=m, j=n;
    while( i>0 && j>0){
```

```
if ( a[i-1] == b[j-1]) {
    lcs.push_back(a[i-1]);
    i--, j--;
}
else if( dp[i-1][j] >= dp[i][j-1]) {
    i--;
}
else{
    j--;
}
reverse(lcs.begin(), lcs.end());
return lcs;
}
```

Set - 04

Algorithm:

(1)

1. Input : list of n activities with start[i], fini
2. Sort activities by ascending order of finis time.
3. Select first activity then repeatedly select next activity whose start time ≥ finish tim of last selected.
4. Output : set of selected activities and coun

(2)

```
struct Activity {
    int id;
    int start, finish;
};
int main() {
    int n;
    cout << "Enter number of activities: ";
    cin >> n;
    vector<Activity> acts(n);
    cout << "Enter start and finish times for each
        activity: \n";
```

```
for(int i=0; i<n; i++){
    cout<<"Activity "<<i+1<<" (start, finish): ";
    cin >> acts[i].start >> acts[i].finish;
    acts[i].id = i+1;
}
sort(acts.begin(), acts.end(), [](const Activity
    & &a, const Activity &b){
        return a.finish > b.finish;
    });
vector<Activity> selected;
int lastFinish = -1;

for(auto &a : acts){
    if(a.start >= lastfinish){
        selected.push_back(a);
        lastFinish = a.finish;
    }
}
cout << "\nMaximum number of non- - - "
    <<selected.size() << endl;
cout << "selected activities (id, start, finish)
    - - - ")
```

```
for(auto &a : acts selected){
    cout << "Activity " << a.id << ": "
    << a.start << " " << a.finish << endl;
}
return 0;
}
```