Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Computer Science & Engineering

Mid Term Report on

SMART BUDGET PLANNER

(0/1 Knapsack Problem Solver)

Project-I

BACHELOR OF TECHNOLOGY
(Computer Science and Engineering)

SUBMITTED BY:

Sagar Jaswal (2330193)

Divyanshi Dadhawal (2330099)

Geetanjali shree (2330104)

Muskan kumari (2420360)

Apr 2025

Under the Guidance of

Dr.Annu
Assistant Professor

Department of Computer Science & Engineering Chandigarh Engineering College Jhanjeri, Mohali - 140307

Table of Contents

S.No	Contents	Page No
1.	Introduction	3
2.	System Requirement	4
3.	Software Requirement Analysis	5-6
4.	Software Design	6-7
5.	Implementation	7-16
6.	References	17

Introduction

The Smart Budget Planner is a frontend web application designed to help users allocate their limited budget in the most optimal way. The system is based on the 0/1 Knapsack Algorithm, a well-known Dynamic Programming technique used for resource optimization problems. The idea is to help users (such as students, investors, or anyone planning a budget) make the best use of their available funds by selecting items or resources that provide the highest value within a fixed budget.

In today's world, managing finances effectively is a key skill. This project simulates a real-world budgeting problem where each item has a cost and value, and the goal is to select the most valuable set of items under

a budget constraint. The algorithm used ensures the most optimal result is achieved through calculation and logic, rather than trial-and-error.

Objectives of the Project

- To develop a web-based tool that assists users in choosing the best combination of resources under a fixed budget.
- To implement and demonstrate the application of the 0/1 Knapsack Algorithm using JavaScript.
- To create an interactive and visually appealing user interface for a better user experience.
- To understand the real-life usage of algorithmic solutions in financial planning.

Tools and Technologies Used / Learnt •

Languages: HTML, CSS, JavaScript

- Concepts Learnt:
  - Frontend web development ○ DOM manipulation
  - Dynamic Programming (0/1 Knapsack Problem) ○
  - Problem-solving through algorithms ○ UI/UX design
  - principles.

Chapter 2: System Requirements

This chapter outlines the software and hardware requirements for the development and usage of the Smart Budget Planner web application. It also includes the basic system flow, data flow, and architecture of the project

2.1 Client-Side (User) Requirements:

- Web Browser: Google Chrome, Mozilla Firefox, Microsoft Edge (latest versions) • Internet Connection: Stable connection for interacting with the server
- Device: ○ Laptop/Desktop (recommended) ○ Minimum screen resolution: 1280x720
- OS Compatibility: Cross-platform (Windows/Linux/macOS)

2.2 Server-Side Requirements:

- Operating System: Windows/Linux server
- Web Server: Apache/Nginx (if deploying manually)
- Backend Language: C++ (for Knapsack logic, compiled as CGI or integrated with backend stack if applicable)
- Database (if applicable):
  - MongoDB / MySQL / PostgreSQL (if storing user data or stock info)
- Storage: 1 GB (minimum for hosting source code, assets, and output logs)
- RAM: Minimum 2 GB for local server testing; 4 GB+ for production

2.3 Development Tools Used:

- Frontend: HTML, CSS, JavaScript
- Backend: JavaScript (Knapsack algorithm logic)
- IDE: Visual Studio Code / Sublime Text / Any code editor • Version Control: Git & GitHub (for collaboration and backup)

2.4 Functional Requirements:

- Allow users to input:
  - Resources names, costs, Value, and budget
- Display:
  - Maximum return possible

Chapter 3: Software Requirement Analysis

3.1 Problem Definition

In real-world budgeting scenarios, individuals often face the challenge of selecting the most valuable combination of resources or investments under a limited budget. This problem is common among students managing allowances, investors distributing funds, or households planning monthly expenses.

The Smart Budget Planner aims to solve this by using the 0/1 Knapsack Algorithm, which helps users select the most optimal set of items (each with a cost and value) such that the total value is maximized without exceeding the budget.

3.2 Modules and Their Functionalities

The project is divided into the following key modules:

---

1. Input Module

- Functionality:
  - Takes user input for:
    - Total budget
    - List of items with their respective cost and value •

    Validation: Ensures proper numeric input and no empty fields.

---

2. Processing Module (Knapsack Logic)

- Functionality:
  - Applies the 0/1 Knapsack Algorithm to calculate:
    - Maximum value achievable within the budget
    - Optimal subset of items to select
- Algorithm Used: Dynamic Programming implementation of the Knapsack Problem.

---

3. Output Module

- Functionality:
  - Displays:
    - Selected items

- Total cost and total value
- Unused budget (if any) ○ Shows real-time results after computation.

---

4. UI/UX Module

- Functionality:
  - Provides a clean and user-friendly interface ○ Uses HTML/CSS for layout and styling ○ Ensures the app is responsive and visually appealing

---

5. Visualization Module

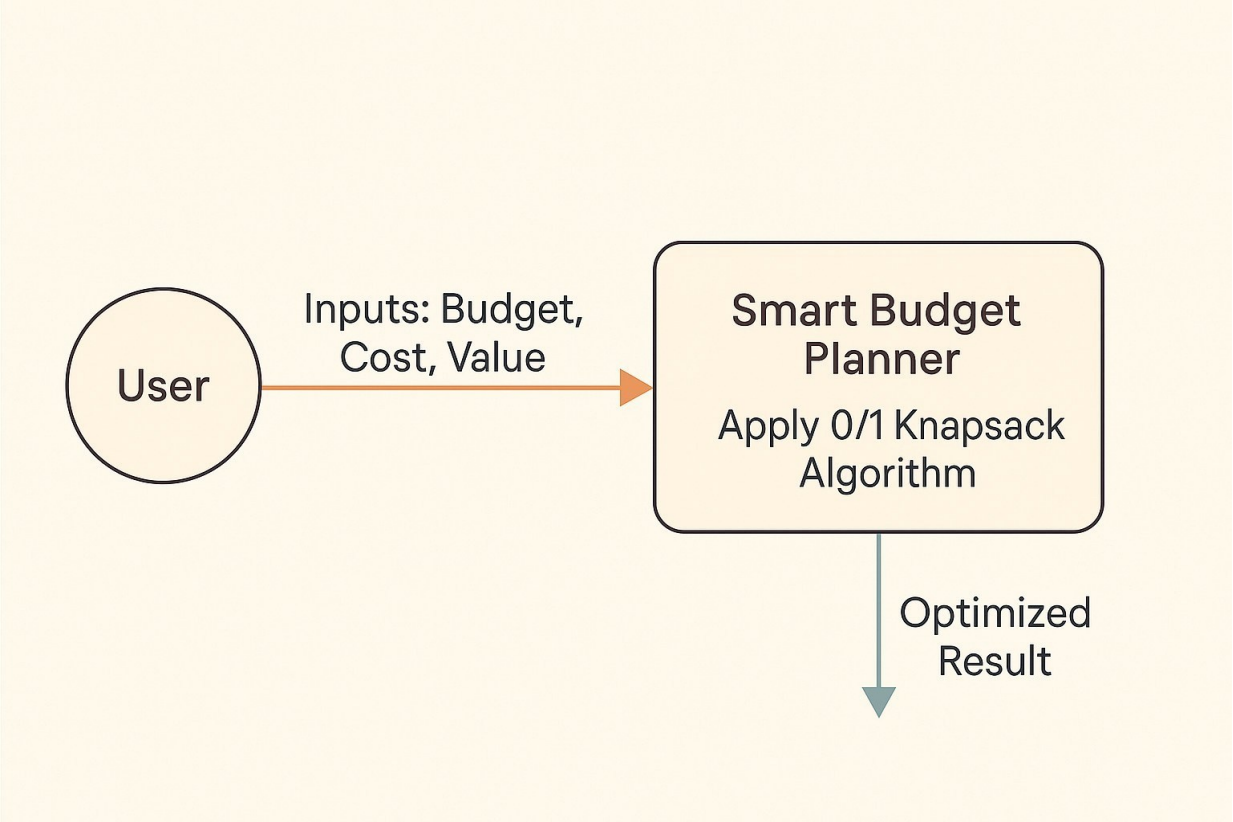
- Functionality:
  - Graphical representation of:
    - Items selected vs. not selected
    - Budget distribution ○ Can use bar charts or pie charts for better understanding. **Chapter 4: Software Design**

4.1 Overview

The *Smart Budget Planner* has been designed using a modular and user-centric approach. Since it is a frontend-only application, all processing is done on the client side using JavaScript. However, the system is designed to be scalable so that backend/database functionality can be added in the future if needed.

4.2 Data Flow Diagrams (DFDs)

Since no backend/database is currently used, the DFD represents how the input data flows through the system.



Relationships:

- One user can have multiple budget plans
- Each budget plan can contain multiple items

Chapter 5: Implementation

5.1 Overview

The *Smart Budget Planner* is developed using HTML, CSS, and JavaScript. The project contains two main web pages:

1. Home Page (index.html) – For navigation and introduction
2. Planner Page (planner.html) – Where the user enters input and gets optimized output

All processing is done in the browser using JavaScript and the 0/1 Knapsack Algorithm.

5.2 File Structure

├── index.html ← Home Page
├── planner.html ← Budget Planning Page
├── style.css ← CSS Styling File
├── script.js ← JavaScript Logic File
└── assets/ ← (Optional images/icons)

5.3 Code Outline and Key Snippets

5.3.1. HTML Code1 (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Smart Budget Planner</title>
  <link rel="stylesheet" href="css/style.css" />
</head>
<body>
  <div class="navbar">
    <div class="logo">💼 Smart Budget Planner</div>
    <div class="nav-links">
      <a href="#">Features</a>
```

```
    <a href="#">About</a>
    <a href="#">Help</a>
   </div>
   <div class="login">
    <a href="#">Login</a>
   </div>
  </div>
  <div class="hero">
   <h1>Plan Smarter, Allocate Better</h1>
   <p>Input your budget, resources, and check how you can maximize value using smart allocation.</p>
   <table class="info-table">
    <tr><th>Step</th><th>What to Do</th></tr>
    <tr><td>1</td><td>Enter total budget</td></tr>
    <tr><td>2</td><td>Add items with cost and value</td></tr>
<tr><td>3</td><td>Click “Check Value and Cost” to proceed</td></tr> </table>
   <a href="planner.html" class="cta-button">Check Value and Cost</a>
  </div>
</body>
</html>
```

5.3.2. HTML Code2 (Planner.html)

```
 <!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Smart Budget Planner</title>
  <link rel="stylesheet" href="css/style.css" />
</head>
```

```
  <body class="planner-body">
    <div class="container">
      <h2>Resource Allocation</h2>
      <div class="form-section">
<input type="number" id="budget" placeholder="Enter Total Budget" />
        <input type="text" id="name" placeholder="Resource Name" />
        <input type="number" id="cost" placeholder="Resource Cost" />
        <input type="number" id="value" placeholder="Resource Value" />
        <button onclick="addResource()">Add Resource</button>
      </div>
      <div class="resource-section">
        <h3>Resources</h3>
        <table id="resourceTable">
    <tr><th>Name</th><th>Cost</th><th>Value</th></tr>
        </table>
        <button onclick="allocateResources()">Allocate Resources</button>
      </div>
      <div class="result-section">
        <h3>Selected Resources</h3>
        <ul id="selectedList"></ul>
        <p>Total Value: <span id="totalValue">0</span></p>
        <p>Total Cost: <span id="totalCost">0</span></p>
      </div>
    </div>
    <script src="js/script.js"></script>
  </body>
  </html>
```

5.3.3. CSS(Style.css)

```
body {  margin: 0; fontfamily:
 'Segoe UI', sansserif;
 background:
url('https://images.unsplash.c om/photo-1522202176988-
66273c2fd55f') no-repeat
center center/cover;   color:
#333; } .navbar {
background-color:
#ffffffcc;     display:  flex;
justify-content:
spacebetween;    padding:
15px 30px;   align-items:
center;   position: sticky;
top: 0; } .logo {   fontweight:
bold;   font-size:
1.5em;
}  .nav-links a, .login a
{   margin: 0 10px;
textdecoration: none;
color: #2563eb;
} .hero {   text-align: center;   padding:
60px 20px;   backgroundcolor:
rgba(255,255,255,0.9);
}  .info-table
{   margin:
20px auto;
bordercollaps
e: collapse;
```

```
} .info-table th,
.infotable td { border:
1px solid #ccc;
padding: 10px; } .cta-
button { display:
inline-block;
background-color:
skyblue; color: white;
padding: 12px 24px;
margin-top: 20px;
text-decoration: none;
border-radius: 8px; }
.cta-button:hover {
background-color:
white; color: skyblue;
border: 1px solid
skyblue; }
.plannerbody {
backgroundcolor:
#f4f6f8; padding:
20px; } .container {
maxwidth: 900px;
margin: auto;
background: white;

 padding: 30px; border-radius:
10px; box-shadow: 0 0 10px
rgba(0,0,0,0.1); } input, button
{ margin: 10px 0; padding:
10px; width: 100%; border-
radius: 8px; } table { width:
```

```
100%;   margin-top: 10px;
border-collapse: separate;
borderspacing: 0 8px; } th, td {
text-align: center;   padding:
10px;   backgroundcolor:
#f9fafb;   border: 1px solid #ccc;
border-radius:

6px; } .result-section
{   margin-top: 20px;
}
```

5.4.4. JS Code (Script.js)

```
const resources = [];  function addResource() {   const name
= document.getElementById('name').value;    const cost =
parseInt(document.getElementById('cost').value);       const
value = parseInt(document.getElementById('value').value);


 if (!name || isNaN(cost) || isNaN(value))
{    alert('Please enter valid details.');
return;
 }


 resources.push({ name, cost, value });   updateTable();


 document.getElementById('name').value = '';
document.getElementById('cost').value = '';
document.getElementById('value').value = '';
}
```

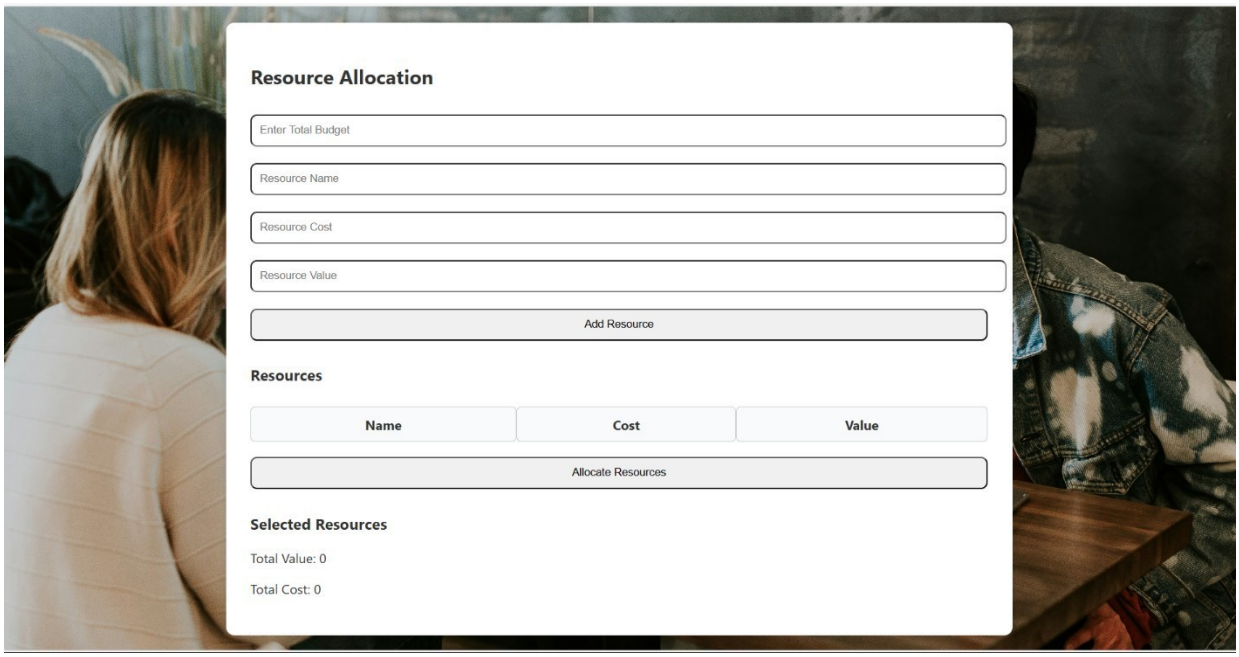
```
function updateTable() {  const table = document.getElementById('resourceTable');
table.innerHTML =
'<tr><th>Name</th><th>Cost</th><th>Value</th></tr>';

 for (let res of resources) {  const row = table.insertRow();  row.innerHTML =
`<td>${res.name}</td><td>${res.cost}</td><td>${res.value}</td>`;

 }
}

function allocateResources() {  const budget =
parseInt(document.getElementById('budget').value);  if
(isNaN(budget)) {  alert('Please enter a valid budget.');  return;
 }

 const n = resources.length;  const W = budget;  const dp =
Array.from({ length: n + 1 }, () => Array(W + 1).fill(0));

 for (let i = 1; i <= n; i++) {  for (let w = 0; w <= W;
w++) {  if (resources[i - 1].cost <= w) {  dp[i][w] =
Math.max(  resources[i - 1].value + dp[i - 1][w -
resources[i - 1].cost],  dp[i - 1][w]
  );
 } else {
dp[i][w] = dp[i -
1][w];
 }
 }
 }
```

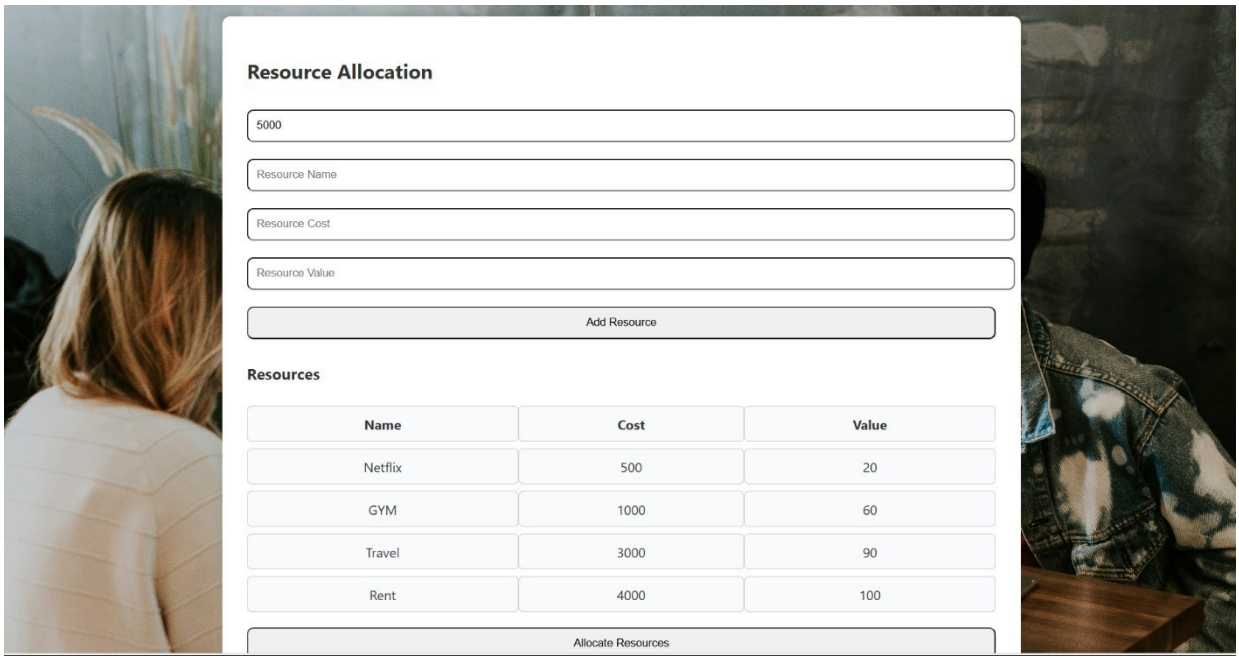
```
let w = W;    let
selected = [];


 for (let i = n; i > 0 && w >= 0; i--)
{    if (dp[i][w] !== dp[i -
1][w])                              {
selected.push(resources[i  -  1]);
w -= resources[i - 1].cost;
   }
 }


 const ul = document.getElementById('selectedList');   ul.innerHTML =
";   let totalCost = 0;   for (let item of selected) {     const li =
document.createElement('li');     li.textContent = `${item.name} (Cost:
${item.cost}, Value: ${item.value})`;     ul.appendChild(li);     totalCost +=
item.cost;
 }


 document.getElementById('totalValue').textContent = dp[n][W];
document.getElementById('totalCost').textContent = totalCost;
}
```

5.4 Future Scope for Backend Integration

Though currently frontend-only, the project is structured to allow backend integration in the future. This can include:

- Saving user data in a SQL database
- Using stored procedures for optimized queries
- Creating REST APIs to interact with the frontend





the user to enter the total budget and add multiple items with cost and value

REFERENCES

1) Knapsack Dynamic Programming Approach **:-** *Knapsack Problem :- https://en.wikipedia.org/wiki/Knapsack_problem*

2) Risk-Adjusted Returns : *Risk-Adjusted Return On Capital https://en.wikipedia.org/wiki/Risk-adjusted_return_on_capital*

3) The Gap Between Algorithmic Finance And Individual Investment Strategies **:** *Algorithmic trading* **:-** *https://en.wikipedia.org/wiki/Algorithmic_trading*

4) Knapsack Dynamic Programming: **GeeksForGeeks:** *Geeksforgeeks webpage (n.d.). 0/1 Knapsack Problem* https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

5) *dynamic programming:* ***GeeksForGeeks:*** *Geeksforgeeks webpage (n.d.). Dynamic Programming or DP: https://www.geeksforgeeks.org/dynamic-programming/*

6) *Linear Programming* **:** ***GeeksForGeeks:*** *Geeksforgeeks webpage (n.d.). Linear Programming | geeksforgeeks:* *https://www.geeksforgeeks.org/linear-programming/*

7) ***HTML****: MDN Web Docs. (n.d.). HTML: Hypertext Markup Language. Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/HTML*

8) ***CSS****: MDN Web Docs. (n.d.). CSS: Cascading Style Sheets. Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/CSS*

9) ***JavaScript****: MDN Web Docs. (n.d.). JavaScript: The Programming Language of the Web. Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/JavaScript*

10) ***Visual Studio Code (VS Code):*** *Official site of VS code (n.d.). Visual Studio Code documentation: https://code.visualstudio.com/docs*

11) ***Git:*** *Official site of Git (n.d.). Git Documentation: https://git-scm.com/doc*

12) ***GitHub:*** *Official site of GitHub (n.d.). GitHub Documentation (GitHub Docs)*