[@KLWKS_BOT] THANOS

Date of the Session:___/___/_____ **Time of the Session:______to_______**

EX – 11 Solving Problems using Branch and Bound technique

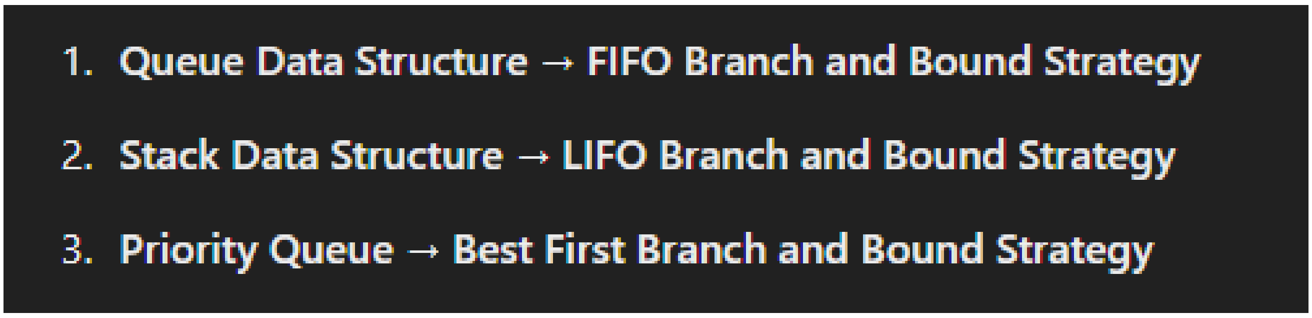
Prerequisites:

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays.

Pre-Lab:

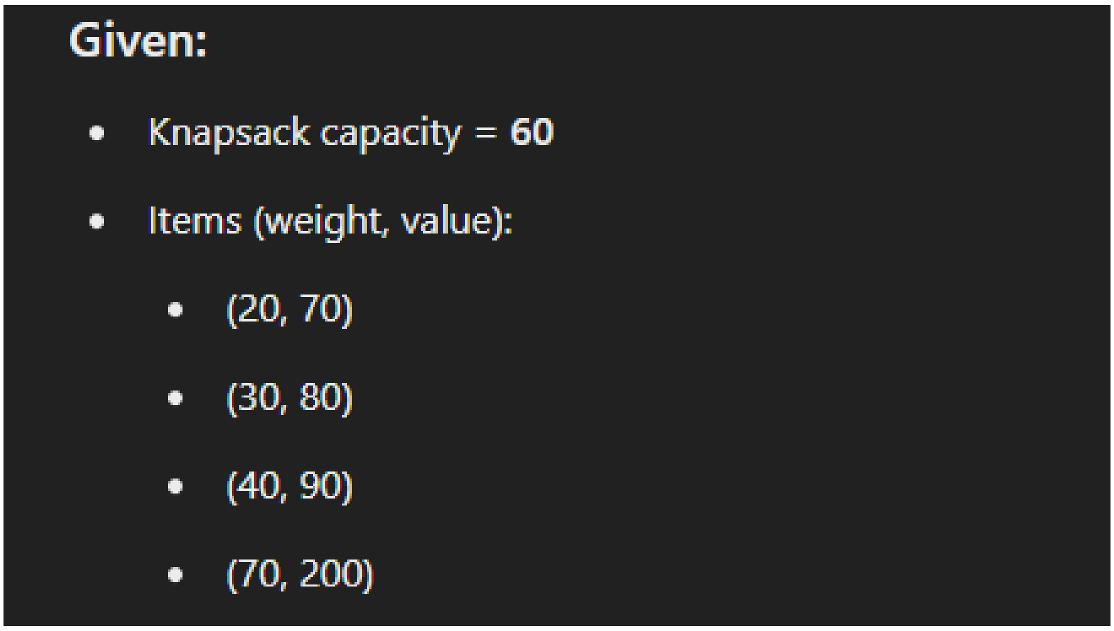
1) Match the data structure used to generate branch and bound strategy.

Queue Data Structure	LIFO Branch and Bound Strategy
Stack Data Structure	Best First branch and bound strategy
Priority Queue	FIFO Branch and Bound Strategy



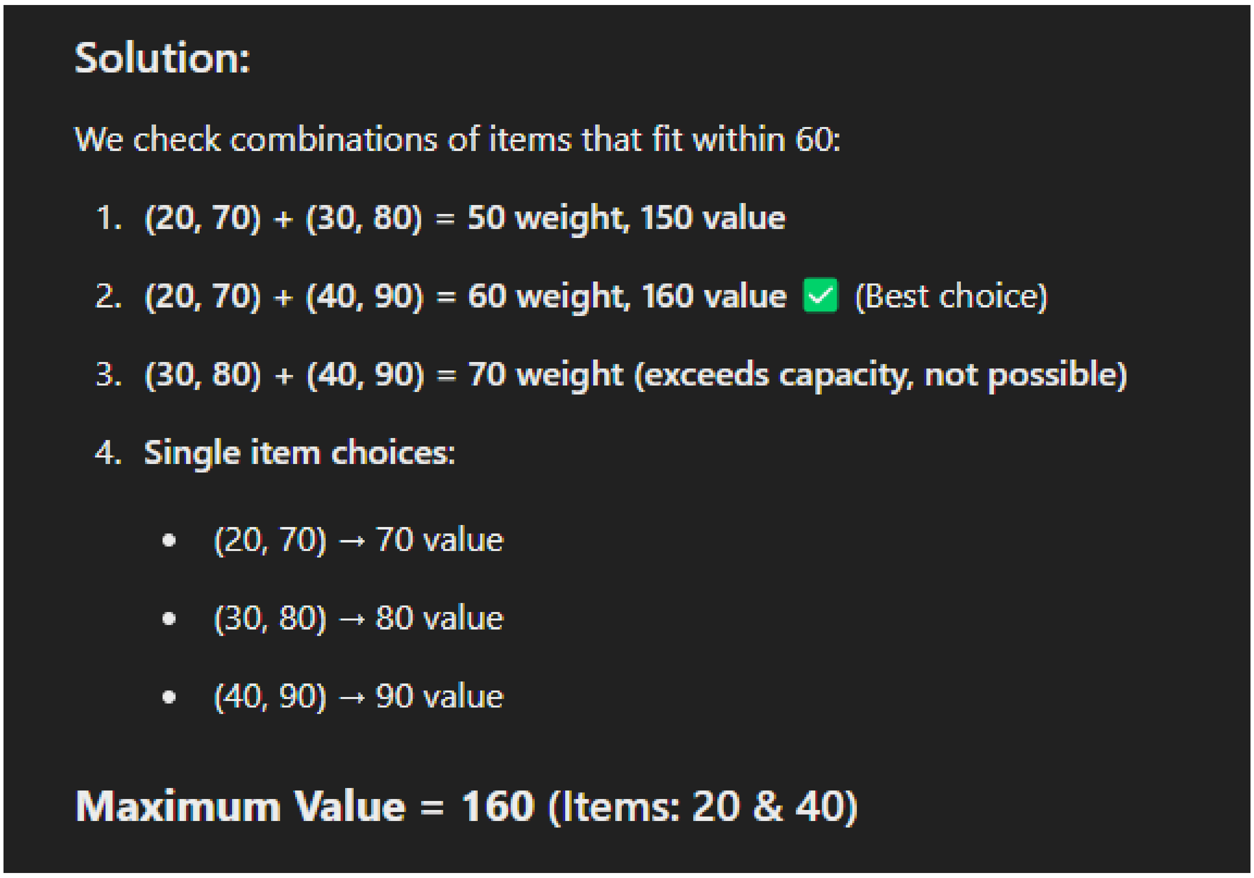
1. Queue Data Structure → FIFO Branch and Bound Strategy
2. Stack Data Structure → LIFO Branch and Bound Strategy
3. Priority Queue → Best First Branch and Bound Strategy

2) You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?



Given:

- Knapsack capacity = **60**
- Items (weight, value):
  - (20, 70)
  - (30, 80)
  - (40, 90)
  - (70, 200)



Solution:

We check combinations of items that fit within 60:

1. **(20, 70) + (30, 80) = 50 weight, 150 value**
2. **(20, 70) + (40, 90) = 60 weight, 160 value** ✅ (Best choice)
3. **(30, 80) + (40, 90) = 70 weight (exceeds capacity, not possible)**
4. **Single item choices:**
   - (20, 70) → 70 value
   - (30, 80) → 80 value
   - (40, 90) → 90 value

Maximum Value = 160 (Items: 20 & 40)

[@KLWKS_BOT] THANOS

In-Lab :

1) A Professor is teaching about Binary Numbers. He wants to know all the possible binary numbers of a given length. Help the Professor to generate all the set of binary strings of length N in ascending order using Branch and Bound Technique.

Input

N = 3

Output

```
000
001
010
011
100
101
110
111
```

Explanation:
Numbers with 3 binary digits are
0, 1, 2, 3, 4, 5, 6, 7

Source code:

```
#include <stdio.h>

void generateBinary(int n, char *binary, int index) {
   if (index == n) {
      binary[index] = '\0';
      printf("%s\n", binary);
      return;
   }

   binary[index] = '0';
   generateBinary(n, binary, index + 1);

   binary[index] = '1';
   generateBinary(n, binary, index + 1);
}

int main() {
   int N = 3;
   char binary[N + 1];
   generateBinary(N, binary, 0);
   return 0;
}
```

[@KLWKS_BOT] THANOS

2) You are in a supermarket shopping for fruits with huge discounts. There are N varieties of fruits available each with a weight C and discount P. Now, select K fruits in a way that maximizes the discount. You only have a bag which can carry a weight of W.

Input
N = 5
P [] = {2, 7, 1, 5, 3}C [] = {2, 5, 2, 3, 4}, W = 8, K = 2.
Output
12
Explanation:
Here, the maximum possible profit is when we take 2 items: item2 (P[1] = 7 and C[1] = 5) and item4 (P[3] = 5 and C[3] = 3).
Hence, maximum profit = 7 + 5 = 12

Source code:

```
#include <stdio.h>

int maxDiscount(int N, int P[], int C[], int W, int K) {
   int dp[K + 1][W + 1];
   for (int i = 0; i <= K; i++)
      for (int j = 0; j <= W; j++)
         dp[i][j] = 0;

   for (int i = 0; i < N; i++) {
      for (int k = K; k > 0; k--) {
         for (int w = W; w >= C[i]; w--) {
            dp[k][w] = (dp[k][w] > dp[k - 1][w - C[i]] + P[i]) ? dp[k][w] : (dp[k - 1][w - C[i]] + P[i]);
         }
      }
   }
   return dp[K][W];
}

int main() {
   int N = 5;
   int P[] = {2, 7, 1, 5, 3};
   int C[] = {2, 5, 2, 3, 4};
   int W = 8;
   int K = 2;

   int result = maxDiscount(N, P, C, W, K);
   printf("%d\n", result);
   return 0;
}
```

[@KLWKS_BOT] THANOS

Post-Lab:

1) Given an array of positive elements, you must flip the sign of some of its elements such that the resultant sum of the elements of array should be minimum non-negative (as close to zero as possible). Return the minimum no. of elements whose sign needs to be flipped such that the resultant sum is minimum non-negative. Note that the sum of all the array elements will not exceed 104.

Input
arr [] = {15, 10, 6}
Output
1
Here, we will flip the sign of 15
and the resultant sum will be 1.

Source code:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SUM 104

int minimum_flips(int arr[], int n) {
   int total_sum = 0;
   for (int i = 0; i < n; i++) {
      total_sum += arr[i];
   }

   int target = total_sum / 2;
   int dp[target + 1];
   for (int i = 0; i <= target; i++) {
      dp[i] = 0;
   }
   dp[0] = 1;

   for (int i = 0; i < n; i++) {
      for (int j = target; j >= arr[i]; j--) {
         if (dp[j - arr[i]]) {
            dp[j] = 1;
         }
      }
   }

   int closest_sum = 0;
   for (int i = target; i >= 0; i--) {
      if (dp[i]) {
         closest_sum = i;
```

[@KLWKS_BOT] THANOS

```
            break;
        }
    }

    int closest_sum_to_zero = total_sum - 2 * closest_sum;
    int flips = 0;
    int remaining_sum = closest_sum;

    int sorted_arr[n];
    for (int i = 0; i < n; i++) {
        sorted_arr[i] = arr[i];
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sorted_arr[i] < sorted_arr[j]) {
                int temp = sorted_arr[i];
                sorted_arr[i] = sorted_arr[j];
                sorted_arr[j] = temp;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if (remaining_sum >= sorted_arr[i]) {
            remaining_sum -= sorted_arr[i];
            flips++;
        }
        if (remaining_sum == 0) {
            break;
        }
    }

    return flips;
}

int main() {
    int arr[] = {15, 10, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = minimum_flips(arr, n);
    printf("Minimum flips needed: %d\n", result);

    return 0;
}
```

[@KLWKS_BOT] THANOS

2) There are some processes that need to be executed. The amount of load that process causes a server that runs it, is being represented by a single integer. The total load caused on a server is the sum of the loads of all the processes that run on that server. You have at your disposal two servers, on which the mentioned processes can be run. Your goal is to distribute given processes between those two servers in a way that, the absolute difference of their loads will be minimized.
Given an array of A[] of N integers, which represents loads caused by successive processes, the task is to print the minimum absolute difference of server loads.

Input

A[] = {1, 2, 3, 4, 5}

Output

1

Explanation:

Distribute the processes with loads {1, 2, 4} on the first server and {3, 5} on the second server, so that their total loads will be 7 and 8, respectively.

The difference of their loads will be equal to 1.

Source code:

```
#include <stdio.h>
#include <stdlib.h>

int minAbsDifference(int A[], int N) {
   int totalLoad = 0;
   for (int i = 0; i < N; i++) {
      totalLoad += A[i];
   }

   int halfLoad = totalLoad / 2;
   int dp[halfLoad + 1];
   for (int i = 0; i <= halfLoad; i++) {
      dp[i] = 0;
   }

   for (int i = 0; i < N; i++) {
      for (int j = halfLoad; j >= A[i]; j--) {
         dp[j] = dp[j] > dp[j - A[i]] + A[i] ? dp[j] : dp[j - A[i]] + A[i];
      }
   }

   return totalLoad - 2 * dp[halfLoad];
}
```

[@KLWKS_BOT] THANOS

```
int main() {
   int A[] = {1, 2, 3, 4, 5};
   int N = sizeof(A) / sizeof(A[0]);
   int result = minAbsDifference(A, N);
   printf("%d\n", result);
   return 0;
}
```

Comments of the Evaluators (if Any)	Evaluator's Observation Marks Secured:________out of [50]. Signature of the Evaluator Date of Evaluation: