DAA ASSIGNMENT

JOB SEQUENCING :

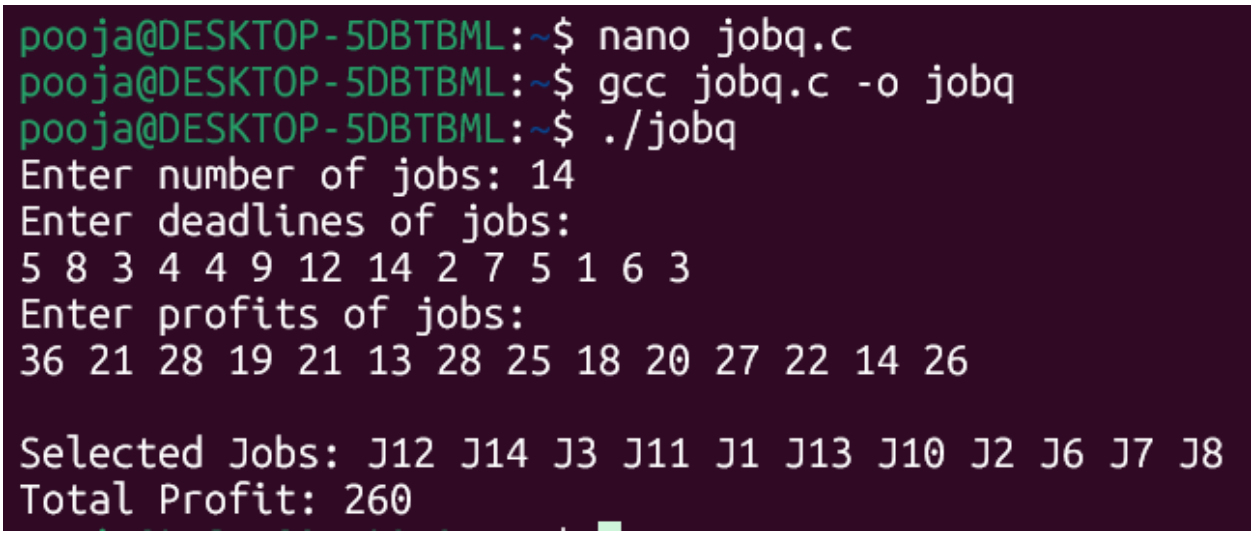
CODE:

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

struct Job {
   int id;
   int deadline;
   int profit;
};
int compare(const void *a, const void *b) {
   struct Job *jobA = (struct Job *)a;
   struct Job *jobB = (struct Job *)b;
   return jobB->profit - jobA->profit;
}

int main() {
   int n, i, j, maxDeadline = 0, totalProfit = 0;

   printf("Enter number of jobs: ");
   scanf("%d", &n);

   struct Job jobs[n];
   printf("Enter deadlines of jobs:\n");
   for (i = 0; i < n; i++) {
```

```
    jobs[i].id = i + 1;
    scanf("%d", &jobs[i].deadline);

    if (jobs[i].deadline > maxDeadline)
        maxDeadline = jobs[i].deadline;
}
printf("Enter profits of jobs:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &jobs[i].profit);
}
qsort(jobs, n, sizeof(struct Job), compare);

int slot[maxDeadline];
for (i = 0; i < maxDeadline; i++)
    slot[i] = -1;
for (i = 0; i < n; i++) {
    for (j = jobs[i].deadline - 1; j >= 0; j--) {
        if (slot[j] == -1) {
            slot[j] = jobs[i].id;   // Store job ID directly
            totalProfit += jobs[i].profit;
            break;
        }
    }
}

printf("\nSelected Jobs: ");
for (i = 0; i < maxDeadline; i++) {
```

```
        if (slot[i] != -1)
            printf("J%d ", slot[i]);
    }

    printf("\nTotal Profit: %d\n", totalProfit);

    return 0;
}
```

RESULT:



HUFFMAN CODE:

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

#define MAX_TREE_HT 50
#define MAX_CHAR 256

struct MinHNode {
  char item;
  unsigned freq;
  struct MinHNode *left, *right;
};

struct MinHeap {
  unsigned size;
  unsigned capacity;
  struct MinHNode **array;
};

int totalBits = 0;
struct MinHNode *newNode(char item, unsigned freq) {
  struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));
  temp->left = temp->right = NULL;
  temp->item = item;
  temp->freq = freq;
  return temp;
```

```
}
struct MinHeap *createMinH(unsigned capacity) {
  struct MinHeap *minHeap = (struct MinHeap *)malloc(sizeof(struct MinHeap));
  minHeap->size = 0;
  minHeap->capacity = capacity;
  minHeap->array = (struct MinHNode **)malloc(capacity * sizeof(struct MinHNode *));
  return minHeap;
}

void swapMinHNode(struct MinHNode **a, struct MinHNode **b) {
  struct MinHNode *t = *a;
  *a = *b;
  *b = t;
}

void minHeapify(struct MinHeap *minHeap, int idx) {
  int smallest = idx;
  int left = 2 * idx + 1;
  int right = 2 * idx + 2;

  if (left < minHeap->size &&
      minHeap->array[left]->freq < minHeap->array[smallest]->freq)
    smallest = left;

  if (right < minHeap->size &&
      minHeap->array[right]->freq < minHeap->array[smallest]->freq)
    smallest = right;
```

```
  if (smallest != idx) {
    swapMinHNode(&minHeap->array[smallest], &minHeap->array[idx]);
    minHeapify(minHeap, smallest);
  }
}

int checkSizeOne(struct MinHeap *minHeap) {
  return (minHeap->size == 1);
}

struct MinHNode *extractMin(struct MinHeap *minHeap) {
  struct MinHNode *temp = minHeap->array[0];
  minHeap->array[0] = minHeap->array[minHeap->size - 1];
  --minHeap->size;
  minHeapify(minHeap, 0);
  return temp;
}

void insertMinHeap(struct MinHeap *minHeap, struct MinHNode *minHeapNode) {
  ++minHeap->size;
  int i = minHeap->size - 1;

  while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
    minHeap->array[i] = minHeap->array[(i - 1) / 2];
    i = (i - 1) / 2;
  }
```

```
  minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap *minHeap) {
  int n = minHeap->size - 1;
  for (int i = (n - 1) / 2; i >= 0; --i)
    minHeapify(minHeap, i);
}

int isLeaf(struct MinHNode *root) {
  return !(root->left) && !(root->right);
}

struct MinHeap *createAndBuildMinHeap(char item[], int freq[], int size) {
  struct MinHeap *minHeap = createMinH(size);

  for (int i = 0; i < size; ++i)
    minHeap->array[i] = newNode(item[i], freq[i]);

  minHeap->size = size;
  buildMinHeap(minHeap);

  return minHeap;
}

struct MinHNode *buildHuffmanTree(char item[], int freq[], int size) {
  struct MinHNode *left, *right, *top;
```

```
  struct MinHeap *minHeap = createAndBuildMinHeap(item, freq, size);

  while (!checkSizeOne(minHeap)) {
    left = extractMin(minHeap);
    right = extractMin(minHeap);

    top = newNode('$', left->freq + right->freq);
    top->left = left;
    top->right = right;

    insertMinHeap(minHeap, top);
  }
  return extractMin(minHeap);
}

void printArray(int arr[], int n) {
  for (int i = 0; i < n; ++i)
    printf("%d", arr[i]);
}

void printHCodes(struct MinHNode *root, int arr[], int top) {
  if (root->left) {
    arr[top] = 0;
    printHCodes(root->left, arr, top + 1);
  }
  if (root->right) {
    arr[top] = 1;
```

```
  printHCodes(root->right, arr, top + 1);
 }

 if (isLeaf(root)) {
  printf("  %c   | ", root->item);
  printArray(arr, top);
  printf("  (Freq=%d, Length=%d)\n", root->freq, top);
  totalBits += root->freq * top;
 }
}

int main() {
 char text[1000];
 int freqTable[MAX_CHAR] = {0};

 printf("Enter a sentence:\n");
 fgets(text, sizeof(text), stdin);

 for (int i = 0; text[i] != '\0'; i++) {
  if (text[i] != '\n')
   freqTable[(int)text[i]]++;
 }

 char chars[MAX_CHAR];
 int freq[MAX_CHAR];
 int size = 0;
```

```
  for (int i = 0; i < MAX_CHAR; i++) {
    if (freqTable[i] > 0) {
      chars[size] = (char)i;
      freq[size] = freqTable[i];
      size++;
    }
  }
  struct MinHNode *root = buildHuffmanTree(chars, freq, size);
  int arr[MAX_TREE_HT], top = 0;
  printf("\nChar | Huffman Code\n");
  printf("---------------------\n");
  printHCodes(root, arr, top);
  printf("\nTotal Encoded Length = %d bits\n", totalBits);
  return 0;
}
```

RESULT:

```
pooja@DESKTOP-5DBTBML:~$ nano huff.c
pooja@DESKTOP-5DBTBML:~$ gcc huff.c -o huff
pooja@DESKTOP-5DBTBML:~$ ./huff
Enter a sentence:
amritavishwavidyapeethamchennaicampus

Char | Huffman Code
--------------------
  a   | 00  (Freq=7, Length=2)
  c   | 0100  (Freq=2, Length=4)
  n   | 0101  (Freq=2, Length=4)
  i   | 011  (Freq=4, Length=3)
  p   | 1000  (Freq=2, Length=4)
  s   | 1001  (Freq=2, Length=4)
  y   | 10100  (Freq=1, Length=5)
  r   | 10101  (Freq=1, Length=5)
  w   | 10110  (Freq=1, Length=5)
  d   | 101110  (Freq=1, Length=6)
  u   | 101111  (Freq=1, Length=6)
  e   | 1100  (Freq=3, Length=4)
  m   | 1101  (Freq=3, Length=4)
  h   | 1110  (Freq=3, Length=4)
  v   | 11110  (Freq=2, Length=5)
  t   | 11111  (Freq=2, Length=5)

Total Encoded Length = 141 bits
```