Code:

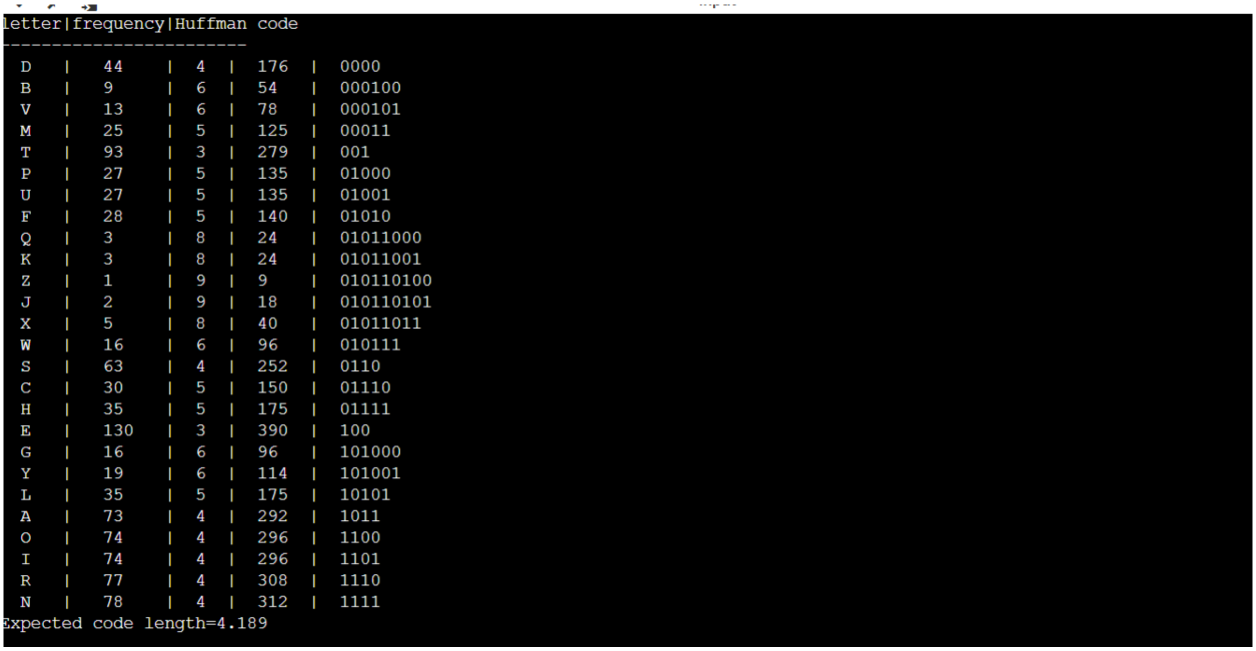
```
// Huffman Coding using minheaps
#include <stdio.h>
#include <stdlib.h>
//caluculating maximum height of huffman tree
#define MAX_TREE_HT 100
//declaring a huffman tree node
struct MinHNode {
  char item;
  unsigned freq;
  struct MinHNode *left, *right;
};
//Minheap-group of huffman tree nodes/min heaps
struct MinHeap {
  unsigned size;
  unsigned capacity;
  struct MinHNode **array;
};
// Create nodes by allotting a min heap with given character and frequency
struct MinHNode *newNode(char item, unsigned freq) {
//creating a min heap temp
  struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));
  temp->left = temp->right = NULL;
  temp->item = item;
  temp->freq = freq;

  return temp;
}

// Create min heap and assigning size,capacity
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(minHeap->capacity * sizeof(struct MinHNode
*));
  return minHeap;
}

// Function to swap two min heap nodes
void swapMinHNode(struct MinHNode **a, struct MinHNode **b) {
  struct MinHNode *t = *a;
  *a = *b;
  *b = t;
}

// Heapify function
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);
  }
}

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

// Extract min value node from heap
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;
}
// Inserting new node to Min heap
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;
}
//function for building minimum heap
void buildMinHeap(struct MinHeap *minHeap) {
  int n = minHeap->size - 1;
  int i;

  for (i = (n - 1) / 2; i >= 0; --i)
    minHeapify(minHeap, i);
}
//function to check whether it is leaf or not
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;
}
//main function that builds the huffman tree
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);
}
// Print the array by transversing
void printArray(int arr[], int n) {
  int i;
  for (i = 0; i < n; ++i)
    printf("%d", arr[i]);

  printf("\n");
}
void spacer(int x){
  if(x>9 && x<100){
          printf(" |");
     }
    if(x<10){
          printf("  |");
     }
     if(x>100){
          printf("|");
     }
}
int ark[121];
//prints huffman codes from the roots of huffman tree
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);
     printf("  %d   ", root->freq);
     int x=root->freq;
     int y=x*top;
     ark[x]=y;
     spacer(x);
     printf("  %d  |",top);
     printf("  %d  ",y);
     spacer(y);
     printf("  ");
    printArray(arr, top);
  }
}

// Wrapper function
void HuffmanCodes(char item[], int freq[], int size) {
  struct MinHNode *root = buildHuffmanTree(item, freq, size);

  int arr[MAX_TREE_HT], top = 0;

  printHCodes(root, arr, top);
}
void printsum(int arr[]){
    int sum=0;
    for(int i=0;i<121;i++){
        sum=sum+arr[i];
    }
    printf("%d",sum);
}
int main() {
//declaring input-I took a random input
  char arr[] = {'A', 'B', 'C', 'D', 'E', 'F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
  int freq[] = {77,17,32,42,120,24,17,50,76,4,7,42,24,67,67,20,5,59,67,85,37,12,22,4,22,2};
//assigning value of size
  int size = sizeof(arr) / sizeof(arr[0]);
  printf("letter|frequency|Huffman code ");
  printf("\n------------------------\n");
  HuffmanCodes(arr, freq, size);
  printsum(ark);
}
```

Test cases:

1-English alphabets as in Moret:

```
    char arr[] = {'A', 'B', 'C', 'D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
    float frq[]=
{0.073,0.009,0.030,0.044,0.130,0.028,0.016,0.035,0.074,0.002,0.003,0.035,0.025,0.078,0.074,
0.027,0.003,0.077,0.063,0.093,0.027,0.013,0.016,0.005,0.019,0.001};
```

Results:



```
letter|frequency|Huffman code
------------------------
  D   |   44    |  4  |  176  |  0000
  B   |   9     |  6  |  54   |  000100
  V   |   13    |  6  |  78   |  000101
  M   |   25    |  5  |  125  |  00011
  T   |   93    |  3  |  279  |  001
  P   |   27    |  5  |  135  |  01000
  U   |   27    |  5  |  135  |  01001
  F   |   28    |  5  |  140  |  01010
  Q   |   3     |  8  |  24   |  01011000
  K   |   3     |  8  |  24   |  01011001
  Z   |   1     |  9  |  9    |  010110100
  J   |   2     |  9  |  18   |  010110101
  X   |   5     |  8  |  40   |  01011011
  W   |   16    |  6  |  96   |  010111
  S   |   63    |  4  |  252  |  0110
  C   |   30    |  5  |  150  |  01110
  H   |   35    |  5  |  175  |  01111
  E   |   130   |  3  |  390  |  100
  G   |   16    |  6  |  96   |  101000
  Y   |   19    |  6  |  114  |  101001
  L   |   35    |  5  |  175  |  10101
  A   |   73    |  4  |  292  |  1011
  O   |   74    |  4  |  296  |  1100
  I   |   74    |  4  |  296  |  1101
  R   |   77    |  4  |  308  |  1110
  N   |   78    |  4  |  312  |  1111
Expected code length=4.189
```

I have taken frequency as int so in place of changing all parts of code i have taken as:

```
int freq[size];
  for(int i=0;i<size;i++){
      freq[i]=1000*frq[i];
  }
```

2-cyrilic symbols:

We have to extract data from table I have done that part using code:

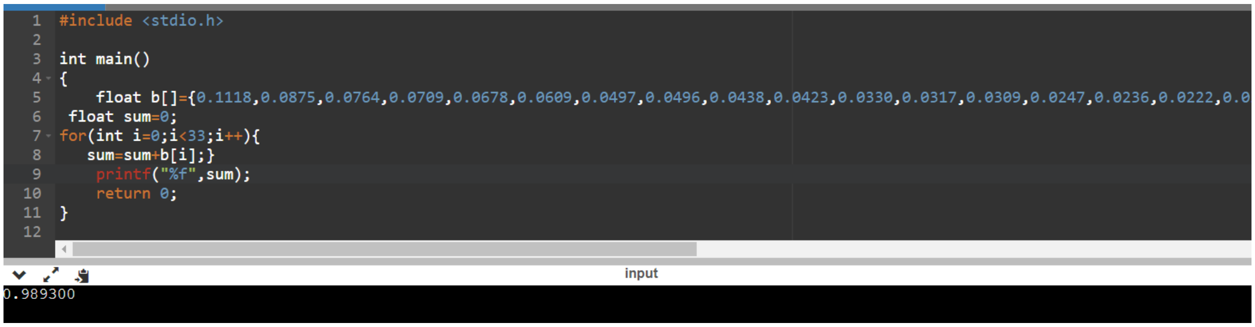
```
#include <stdio.h>

int main()
{
    int a[33];
    char b[33];
    float c[33];
    for(int i=0;i<33;i++){
        scanf("%d",&a[i]);
        scanf("%c",&b[i]);
        scanf("%f",&c[i]);
    }

for(int i=0;i<33;i++){
    printf("%c,",b[i]);}
      printf("\n");
for(int i=0;i<33;i++){
    printf("%f,",c[i]);}
```

return 0;
}

But since the output of symbol if undefined it came as:�
Hence I have done that part manually,
char arr[] = {'О','Е','А','И','Н','Т','С','Л','В','Р','К','М','Д','П','Ы','У','Б','Я','Ь','Г','З','Ч','Й','Ж','Х','Ш','Ю','Ц','Э','Щ','Ф','Ё','Ъ'};
float frq[] = {0.1118,0.0875,0.0764,0.0709,0.0678,0.0609,0.0497,0.0496,0.0438,0.0423,0.0330,0.0317,0.0309,0.0247,0.0236,0.0222,0.0201,0.0196,0.0184,0.0172,0.0148,0.0140,0.0121,0.0101,0.0095,0.0072,0.0047,0.0039,0.0036,0.0030,0.0021,0.0020,0.0002};

Results:

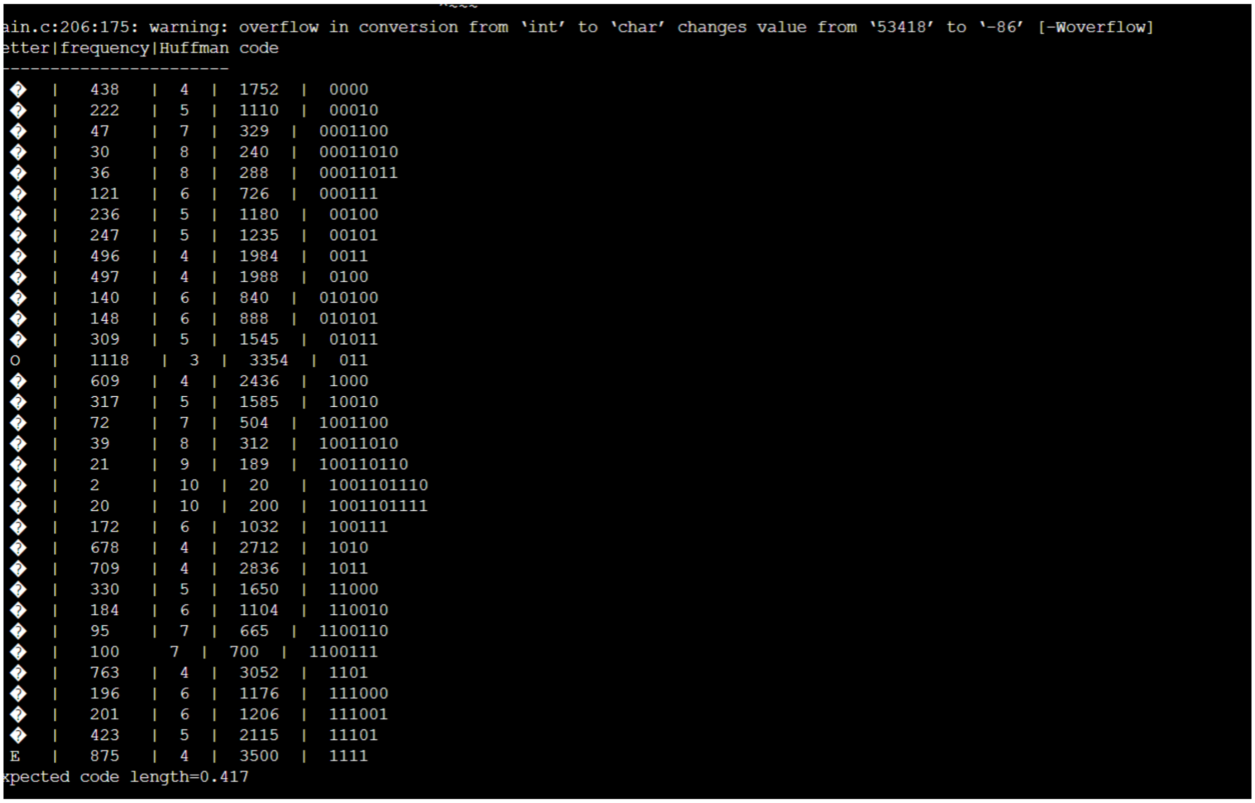
The input of the file are giving error as:
warning: multi-character character constant
This is due to unrecognizable chars in the language used.
Also the total sum of frequency !=1,which is with error of 0.07%



```
#include <stdio.h>

int main()
{
    float b[]={0.1118,0.0875,0.0764,0.0709,0.0678,0.0609,0.0497,0.0496,0.0438,0.0423,0.0330,0.0317,0.0309,0.0247,0.0236,0.0222,0.0
 float sum=0;
for(int i=0;i<33;i++){
   sum=sum+b[i];}
    printf("%f",sum);
    return 0;
}

```

```
0.989300
```

The output of the file is as follows:

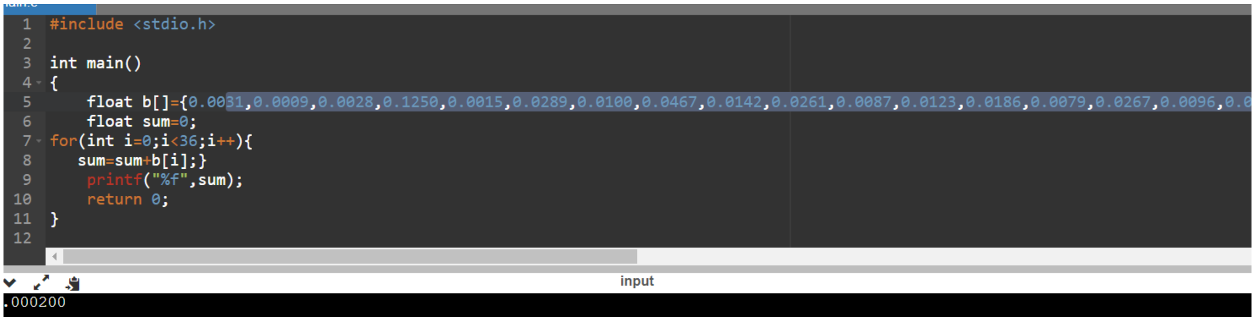


```
                                                                        ^~~~
ain.c:206:175: warning: overflow in conversion from ‘int’ to ‘char’ changes value from ‘53418’ to ‘-86’ [-Woverflow]
etter|frequency|Huffman code
----------------------
 �   |   438    |  4  |  1752  |  0000
 �   |   222    |  5  |  1110  |  00010
 �   |   47     |  7  |  329  |  0001100
 �   |   30     |  8  |  240  |  00011010
 �   |   36     |  8  |  288  |  00011011
 �   |   121    |  6  |  726  |  000111
 �   |   236    |  5  |  1180  |  00100
 �   |   247    |  5  |  1235  |  00101
 �   |   496    |  4  |  1984  |  0011
 �   |   497    |  4  |  1988  |  0100
 �   |   140    |  6  |  840  |  010100
 �   |   148    |  6  |  888  |  010101
 �   |   309    |  5  |  1545  |  01011
 O   |   1118    |  3  |  3354  |  011
 �   |   609    |  4  |  2436  |  1000
 �   |   317    |  5  |  1585  |  10010
 �   |   72     |  7  |  504  |  1001100
 �   |   39     |  8  |  312  |  10011010
 �   |   21     |  9  |  189  |  100110110
 �   |   2      |  10  |  20   |  1001101110
 �   |   20     |  10  |  200  |  1001101111
 �   |   172    |  6  |  1032  |  100111
 �   |   678    |  4  |  2712  |  1010
 �   |   709    |  4  |  2836  |  1011
 �   |   330    |  5  |  1650  |  11000
 �   |   184    |  6  |  1104  |  110010
 �   |   95     |  7  |  665  |  1100110
 �   |   100      7  |  700   |  1100111
 �   |   763    |  4  |  3052  |  1101
 �   |   196    |  6  |  1176  |  111000
 �   |   201    |  6  |  1206  |  111001
 �   |   423    |  5  |  2115  |  11101
 E   |   875    |  4  |  3500  |  1111
xpected code length=0.417
```

length=4.17

Results for arabic symbols:

Error in relative frequency %-0.02%

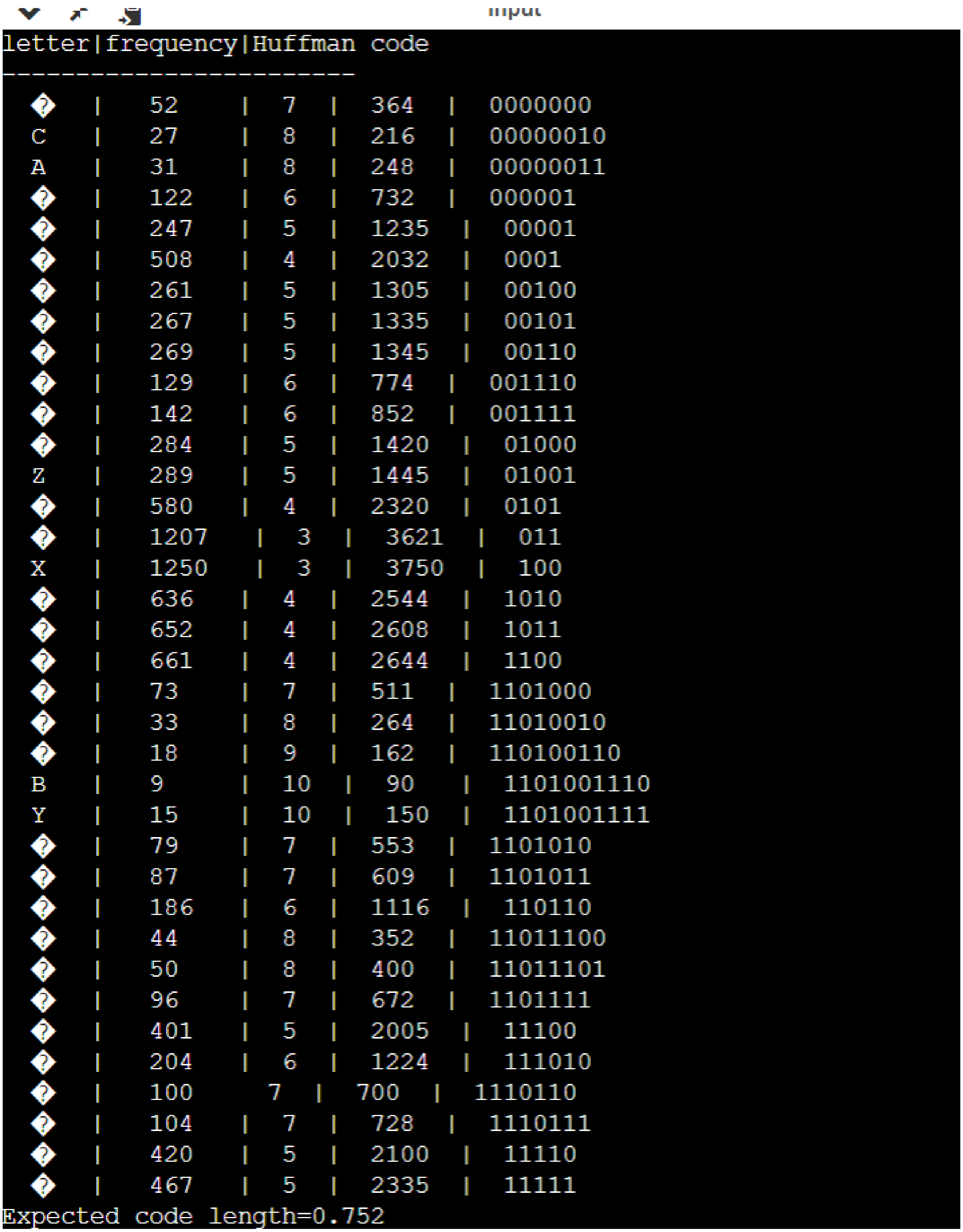


```c
#include <stdio.h>

int main()
{
    float b[]={0.0031,0.0009,0.0028,0.1250,0.0015,0.0289,0.0100,0.0467,0.0142,0.0261,0.0087,0.0123,0.0186,0.0079,0.0267,0.0096,0.0
    float sum=0;
for(int i=0;i<36;i++){
   sum=sum+b[i];}
    printf("%f",sum);
    return 0;
}

```

```
.000200
```

result-1.000200(0.02% error)

Final result:



```
letter|frequency|Huffman code
------------------------
  �    |    52     |  7  |  364   |   0000000
  C    |    27     |  8  |  216   |   00000010
  A    |    31     |  8  |  248   |   00000011
  �    |    122    |  6  |  732   |   000001
  �    |    247    |  5  |  1235   |   00001
  �    |    508    |  4  |  2032   |   0001
  �    |    261    |  5  |  1305   |   00100
  �    |    267    |  5  |  1335   |   00101
  �    |    269    |  5  |  1345   |   00110
  �    |    129    |  6  |  774   |   001110
  �    |    142    |  6  |  852   |   001111
  �    |    284    |  5  |  1420   |   01000
  Z    |    289    |  5  |  1445   |   01001
  �    |    580    |  4  |  2320   |   0101
  �    |    1207    |  3  |  3621   |   011
  X    |    1250    |  3  |  3750   |   100
  �    |    636    |  4  |  2544   |   1010
  �    |    652    |  4  |  2608   |   1011
  �    |    661    |  4  |  2644   |   1100
  �    |    73     |  7  |  511   |   1101000
  �    |    33     |  8  |  264   |   11010010
  �    |    18     |  9  |  162   |   110100110
  B    |    9      |  10  |  90    |   1101001110
  Y    |    15     |  10  |  150   |   1101001111
  �    |    79     |  7  |  553   |   1101010
  �    |    87     |  7  |  609   |   1101011
  �    |    186    |  6  |  1116   |   110110
  �    |    44     |  8  |  352   |   11011100
  �    |    50     |  8  |  400   |   11011101
  �    |    96     |  7  |  672   |   1101111
  �    |    401    |  5  |  2005   |   11100
  �    |    204    |  6  |  1224   |   111010
  �    |    100     7   |  700   |  1110110
  �    |    104    |  7  |  728   |   1110111
  �    |    420    |  5  |  2100   |   11110
  �    |    467    |  5  |  2335   |   11111
Expected code length=0.752
```

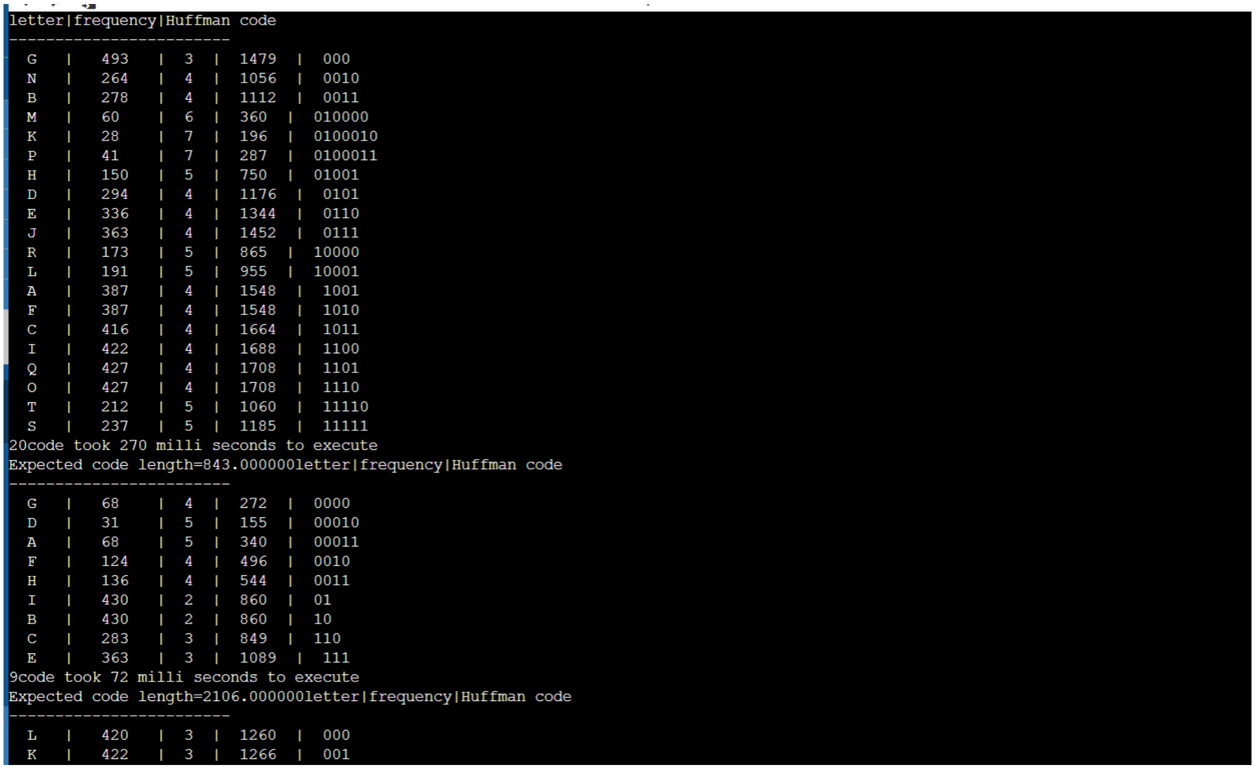
length=7.52

For the second part of the assignment,
Generating sets of n frequencies with increasing n:
Code:

```
int main() {
  for(int i=0;i<10;i++){
  int lower = 1, upper = 36;
    int num = (rand() % (upper - lower + 1)) + lower;
    char* arr;
    int* freq;
   char ar[] = {'A', 'B', 'C', 'D', 'E',
'F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','j'};
   arr = calloc(num, sizeof(char) );
   freq = calloc(num, sizeof(int));
    for(int j=0;j<num;j++){
      arr[j]=ar[j];
    }
 for(int l=0;l<num;l++){
  int lowr = 1, uppr = 500;
    freq[l] = (rand() % (uppr - lowr + 1)) + lowr;
 }

 clock_t t;
  t = clock();
 printf("letter|frequency|Huffman code ");
 printf("\n-----------------------\n");
 HuffmanCodes(arr, freq, num);
  t = clock() - t;
  double time_taken = (((double)t)/CLOCKS_PER_SEC)*1000000;

  printf("%d",num);
   printf("code took %.f milli seconds to execute \n", time_taken);
 printsum(ark);
  }
}
```

By this way,I have allocated arrays dynamically and checked for total time taken for code execution.

Results:



```
letter|frequency|Huffman code
-------------------------
  G   |   493    |  3  |  1479  |  000
  N   |   264    |  4  |  1056  |  0010
  B   |   278    |  4  |  1112  |  0011
  M   |   60     |  6  |  360   |  010000
  K   |   28     |  7  |  196   |  0100010
  P   |   41     |  7  |  287   |  0100011
  H   |   150    |  5  |  750   |  01001
  D   |   294    |  4  |  1176  |  0101
  E   |   336    |  4  |  1344  |  0110
  J   |   363    |  4  |  1452  |  0111
  R   |   173    |  5  |  865   |  10000
  L   |   191    |  5  |  955   |  10001
  A   |   387    |  4  |  1548  |  1001
  F   |   387    |  4  |  1548  |  1010
  C   |   416    |  4  |  1664  |  1011
  I   |   422    |  4  |  1688  |  1100
  Q   |   427    |  4  |  1708  |  1101
  O   |   427    |  4  |  1708  |  1110
  T   |   212    |  5  |  1060  |  11110
  S   |   237    |  5  |  1185  |  11111
20code took 270 milli seconds to execute
Expected code length=843.000000letter|frequency|Huffman code
-------------------------
  G   |   68     |  4  |  272   |  0000
  D   |   31     |  5  |  155   |  00010
  A   |   68     |  5  |  340   |  00011
  F   |   124    |  4  |  496   |  0010
  H   |   136    |  4  |  544   |  0011
  I   |   430    |  2  |  860   |  01
  B   |   430    |  2  |  860   |  10
  C   |   283    |  3  |  849   |  110
  E   |   363    |  3  |  1089  |  111
9code took 72 milli seconds to execute
Expected code length=2106.000000letter|frequency|Huffman code
-------------------------
  L   |   420    |  3  |  1260  |  000
  K   |   422    |  3  |  1266  |  001
```

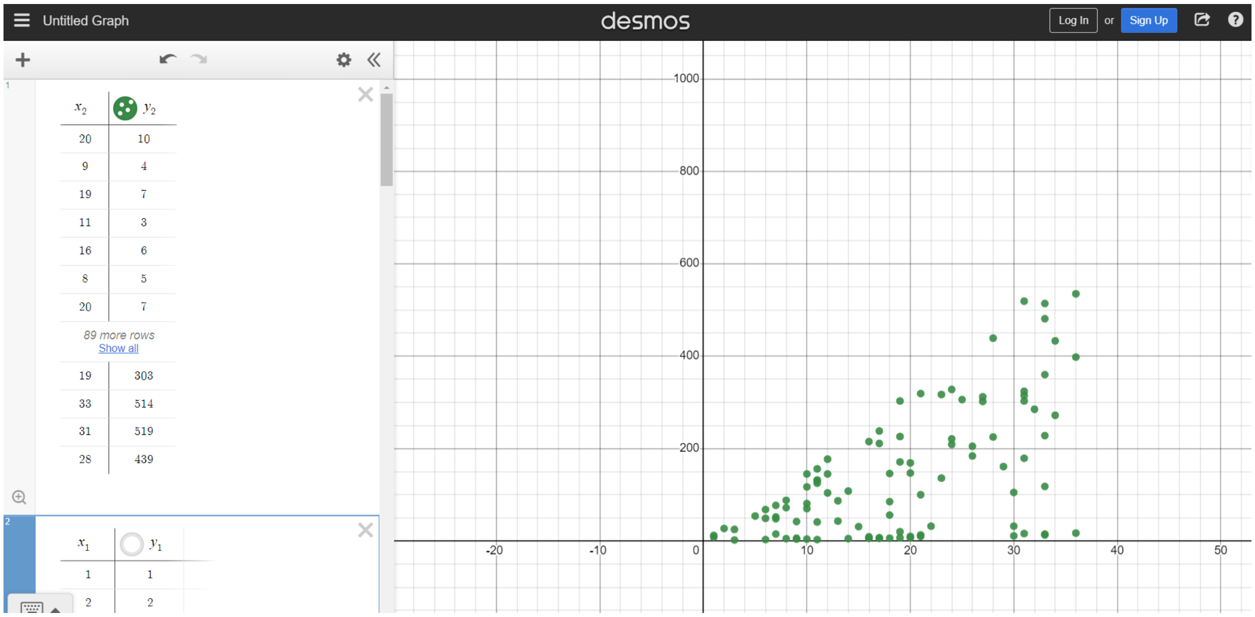
For plotting to graph,the output is reduced to only n,time taken:

1–20,12
9,4
19,9
11,4
16,7
8,7
20,8
17,7
10,4
21,12
17,7
17,10
16,7
30,14
33,18
36,19
33,17
9,8
3,2
18,9

2—20,10
9,3
19,7
11,4
16,5
8,5
20,7
17,6
10,4
21,10
17,5
17,7
16,6
30,11
33,15
36,15
33,12
9,5
3,2
18,6

Graph:

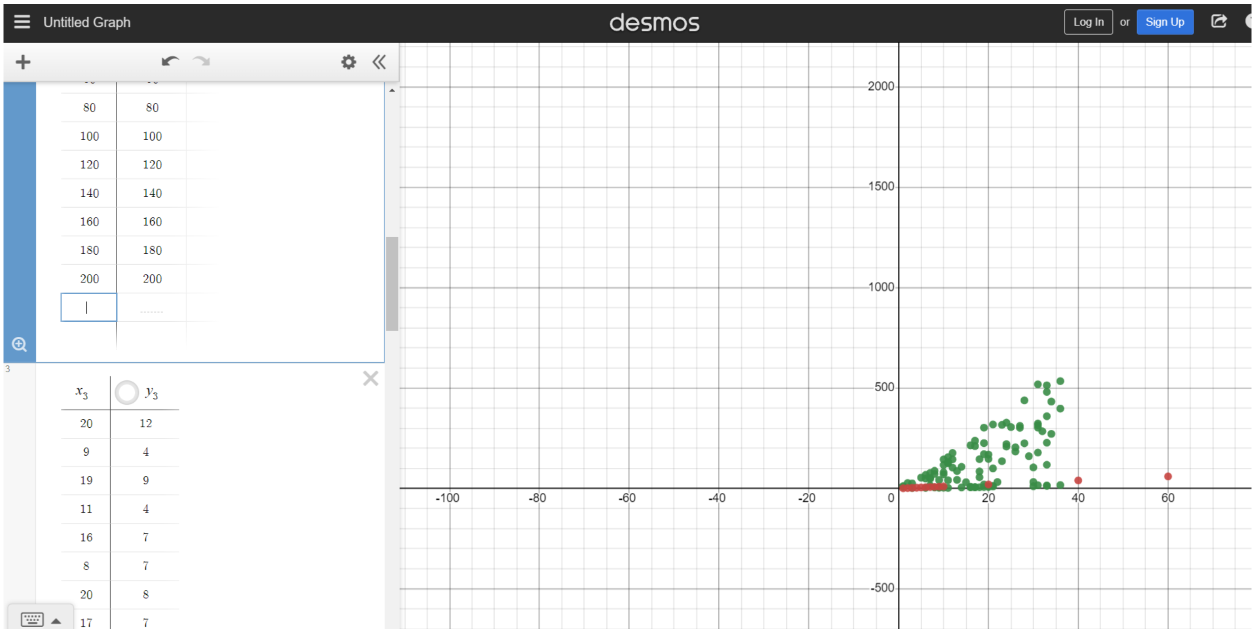
Plot with 200 inputs of both size,time taken



x_2	● y_2
20	10
9	4
19	7
11	3
16	6
8	5
20	7
89 more rows Show all	
19	303
33	514
31	519
28	439

x_1	○ y_1
1	1
2	2

Plot with output comparison with straight line-O(n):

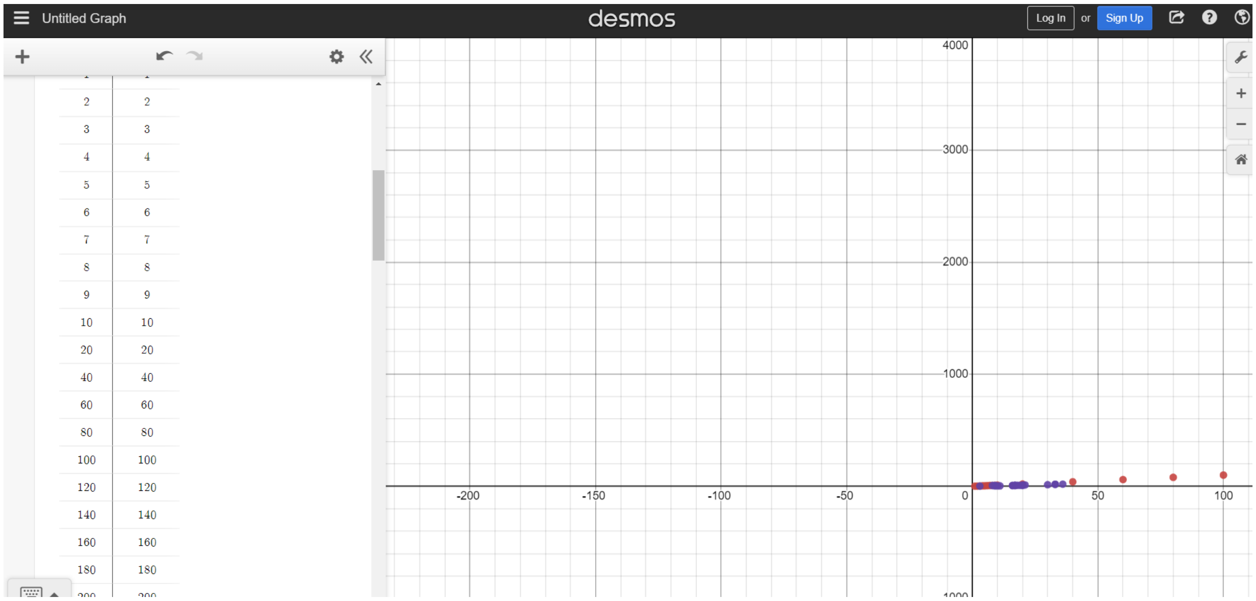


80	80
100	100
120	120
140	140
160	160
180	180
200	200

x_3	y_3
20	12
9	4
19	9
11	4
16	7
8	7
20	8
17	7

With this comparison,we can get to know that the plot is tending to nlogn

Plot with 20 inputs:



2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
20	20
40	40
60	60
80	80
100	100
120	120
140	140
160	160
180	180
200	200

Conclusion:

By these plots,we can conclude that Time complexity of the code tends to **O(nlogn)**