poly.h

```
typedef struct term {
   int coeff;
   int exp;
} term;

typedef struct polynomial {
   int n;
   struct term* t;
} poly;

void init_poly(poly* p, int size);
void append(poly* p, int coeff, int exp);
void display(poly* p);
void add_poly(poly* p1, poly* p2, poly* p3);
void sub_poly(poly* p1, poly* p2, poly* p3);
void quadratic_roots(poly* p);
```

poly.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "poly.h"

void init_poly(poly* p, int size) {
   p->n = 0;
   p->t = (term*) malloc(sizeof(term) * size);
   if(!p-> t) return;
   return;
}; // The pointer is pointing to a dynamically allocated memory with size = sizeof(term) in bytes
```

```
void append(poly* p, int coeff, int exp) {
   p->t[p->n].coeff = coeff;
   p->t[p->n].exp = exp;
   p->n++;
   return;
}; // The current memory address is added with a coefficient and exponent taken front the user and the length of the polynomial is increased by 1


void display(poly* p) {
   int i = 0;
   for(i = 0; i < p->n - 1; i++) {
      printf("%dx^%d + ", p->t[i].coeff, p->t[i].exp);
   };
   printf("%dx^%d", p->t[i].coeff, p->t[i].exp);
   return;
} /**  Each element is iterated over and displayed in an appropriate format
      Time Complexity: O(n)
      Space Complexity: O(1)
**/


void add_poly(poly* p1, poly* p2, poly* p3) { // This considers no repitition of same exponents within a single polynomial
   int i = 0;
   while(i < p1->n) {
      int helper = p1->t[i].exp;
      for(int j = 0; j < p2->n; j++) {
         if(p2->t[j].exp == helper) {
            p3->t[p3->n].coeff = p1->t[i].coeff + p2->t[j].coeff;
            p3->t[p3->n].exp = helper;
            p3->n++;
         };
      };
```

```
        i++;
    };
} /** The exponents of the polynomial are matched using nested for loop and the coefficients are
then added respectively
        Time Complexity: O(n^2)
        Space Complexity: O(n)
**/

void sub_poly(poly* p1, poly* p2, poly* p3) { // This considers no repetition of same exponents
within a single polynomial
    int i = 0;
    while(i < p1->n) {
        int helper = p1->t[i].exp;
        for(int j = 0; j < p2->n; j++) {
            if(p2->t[j].exp == helper) {
                p3->t[p3->n].coeff = p1->t[i].coeff - p2->t[j].coeff;
                p3->t[p3->n].exp = helper;
                p3->n++;
            };
        };
        i++;
    };
}; /** The exponents of the polynomial are matched using a nested for loop and the coefficients are
added respectively
        Time Complexity: O(n^2)
        Space Complexity: O(n)
**/

int isValidQuadratic(poly* p) {
    for(int i = 0; i < p->n; i++) {
        if(p->t[i].exp > 2) {
            return 0;
```

```
    };
  };
  return 1;
}; /** This functions checks if there is any element in the strucutre with exponent greater than 2, which breaks the quadratic of being quadratic */

void quadratic_roots(poly* p) { // standard equation: ax^2 + bx + c = 0
  if(isValidQuadratic(p) == 0) return;

  int a = 0, b = 0, c = 0;
  for(int i = 0; i < p->n; i++) {
    if(p->t[i].exp == 2) {
      a = p->t[i].coeff;
    } else if(p->t[i].exp == 1) {
      b = p->t[i].coeff;
    } else if(p->t[i].exp == 0) {
      c = p->t[i].coeff;
    }
  }
  double discriminant = pow(b, 2) - 4 * a * c;

  if(discriminant < 0) {
    return;
  }

  double root1 = (-b + sqrt(discriminant)) / (2 * a);
  double root2 = (-b - sqrt(discriminant)) / (2 * a);
  printf("%lf %lf\n", root1, root2);

  return;
} /** The roots are calculated using the regular quadratic formula with constant time and space complexity
```

```
    Time Complexity: O(1)
    Space Complexity: O(1)
**/
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "poly.c"

int main() {
    poly* p, *p2, *p3;
    p = (poly*) malloc(sizeof(p));
    p2 = (poly*) malloc(sizeof(p));
    p3 = (poly*) malloc(sizeof(p));
    init_poly(p, 3);
    init_poly(p2, 3);
    init_poly(p3, 3);
    append(p, 4, 2);
    append(p, 7, 1);
    append(p, 3, 0);
    append(p2, 2, 2);
    append(p2, 7, 1);
    add_poly(p, p2, p3);
    sub_poly(p, p2, p3);
    quadratic_roots(p);
    return 0;
}
```

```
PS C:\Users\Aman Morghade\OneDrive\Documents\DSA_CoEP\LabWork1\polynomial> gcc .\main.c
PS C:\Users\Aman Morghade\OneDrive\Documents\DSA_CoEP\LabWork1\polynomial> .\a.exe

-0.750000 -1.000000
PS C:\Users\Aman Morghade\OneDrive\Documents\DSA_CoEP\LabWork1\polynomial>
```