```c
/**
 * Binary Search: the recursive version
 *
 * Created by hengxin on 11/14/21.
 */

#include <stdio.h>
#define LEN 10

int BinarySearch(int key, int dict[], int low, int high);

int main() {
  int dictionary[LEN] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34};

  int key;
  scanf("%d", &key);

  printf("The index of %d is %d.\n", key,
         BinarySearch(key, dictionary, 0, LEN - 1));

  return 0;
}

int BinarySearch(int key, int dict[], int low, int high) {
//  if (low == high) {
//    if (dict[low] == key) {
//      return low;
//    }
//    return -1;
//  }

  if (low > high) {
    return -1;
  }

  int mid = (low + high) / 2;

  if (dict[mid] == key) {
    return mid;
  }

  if (dict[mid] > key) {
    return BinarySearch(key, dict, low, mid - 1);
  }

  return BinarySearch(key, dict, low + 1, high);
}

```

```c
/**
 * file: fib-iter.c
 *
 * Iteratively computing the first n Fibonacci numbers
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>
#include <limits.h>

#define LEN 93
long long fibs[LEN] = {0, 1};

int main() {
  int n;
  scanf("%d", &n);

  for (int i = 2; i < n; i++) {
    fibs[i] = fibs[i - 1] + fibs[i - 2];
  }

  for (int i = 0; i < n; i++) {
    printf("%lld ", fibs[i]);
  }

  printf("\n%lld\n", LLONG_MAX);

  return 0;
}
```

```c
#include <stdio.h>
/**
 * file: fib-iter.c
 *
 * Iteratively computing the first n Fibonacci numbers
 *
 * Created by hengxin on 11/13/21.
*/

int main() {
  int n;
  scanf("%d", &n);

  long long fib1 = 0;
  long long fib2 = 1;
  printf("%lld %lld ", fib1, fib2);

  long long fib3;
  for (int i = 3; i < n; i++) {
    fib3 = fib1 + fib2;
    printf("%lld ", fib3);

    fib1 = fib2;
    fib2 = fib3;
  }

  return 0;
}
```

```c
/**
 * file: fib.c
 *
 * Recursively computing the n-th Fibonacci number
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>

long long Fib(int n);

int main() {
  int n;
  scanf("%d", &n);

  printf("%lld\n", Fib(n));
}

long long Fib(int n) {
  if (n == 0) {
    return 0;
  }

  if (n == 1) {
    return 1;
  }

  return Fib(n - 1) + Fib(n - 2);
}
```

```c
#include <stdio.h>
/**
 * file: gcd-euclid-iter.c
 *
 * Visualization: https://pythontutor.com/c.html#code=int%20main%28%29%20%7B%0A%0A%20%20int%20a%20%3D%2064%3B%0A%20%20int%20b%20%3D%2018%3B%0A%0A%20%20while%20%28a%20!%3D%20b%29%20%7B%0A%20%20%20%20if%20%28a%20%3E%20b%29%20%7B%0A%20%20%20%20%20%20a%20%3D%20a%20-%20b%3B%0A%20%20%20%20%7D%20else%20%7B%0A%20%20%20%20%20%20b%20%3D%20b%20-%20a%3B%0A%20%20%20%20%7D%0A%20%20%7D%0A%20%20%0A%20%20return%200%3B%0A%7D&curInstr=28&mode=display&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D
 *
 * Created by hengxin on 11/14/21.
 */

int main() {
  int a;
  int b;
  scanf("%d %d", &a, &b);

  while (a != b) {
    if (a > b) {
      a = a - b;
    } else {
      b = b - a;
    }
  }

  return 0;
}

```

```c
/**
 * file: gcd-euclid.c
 *
 * Euclid's algorithm:
 *
 * if a > b
 * then gcd(a, b) = gcd(a - b, b)
 * else gcd(a, b) = gcd(a, b - a)
 *
 * Created by hengxin on 11/14/21.
 */

#include <stdio.h>

int GCDEuclid(int a, int b);

int main() {
  int a;
  int b;
  scanf("%d %d", &a, &b);

  printf("gcd(%d, %d) = %d\n", a, b, GCDEuclid(a, b));

  return 0;
}

int GCDEuclid(int a, int b) {
  if (a == b) {
    return a;
  }

  if (a > b) {
    return GCDEuclid(a - b, b);
  }

  if (a < b) {
    return GCDEuclid(a, b - a);
  }
}
```

```c
/**
 * file: gcd.c
 *
 * Iteratively computing the greatest common divisor of two integers.
 *
 * Euclidean algorithm:
 * gcd(a, b) = gcd(b, a % b)
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>

int GCD(int a, int b);

int main() {
  int a = 130;
  int b = 124;

  printf("gcd(%d, %d) = %d\n", a, b, GCD(a, b));

  return 0;
}

int GCD(int a, int b) {
  int tmp;
  while (b != 0) {
    tmp = b;
    b = a % b;
    a = tmp;
  }

  return a;
}
```

```c
/**
 * Recursively computing the greatest common divisor of two integers
 *
 * Euclidean algorithm:
 * gcd(a, b) = gcd(b, a % b)
 *
 * Visualization: https://pythontutor.com/c.html#code=int%20GCD%28int%20a,%20int%20b%29%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20int%20a%20%3D%2064%3B%0A%20%20int%20b%20%3D%2048%3B%0A%0A%20%20printf%28%22gcd%28%25d,%20%25d%29%20%3D%20%25d%5Cn%22,%20a,%20b,%20GCD%28a,%20b%29%29%3B%0A%0A%20%20return%200%3B%0A%7D%0A%0A//%20gcd%28130,%20124%29%20%3D%202%0A//%20gcd%28662,%20414%29%20%3D%202%0Aint%20GCD%28int%20a,%20int%20b%29%20%7B%0A%20%20if%20%28b%20%3D%3D%200%29%20%7B%0A%20%20%20%20return%20a%3B%0A%20%20%7D%0A%0A%20%20return%20GCD%28b,%20a%20%25%20b%29%3B%0A%7D&curInstr=17&mode=display&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>

int GCD(int a, int b);

int main() {
  int a;
  int b;
  scanf("%d %d", &a, &b);

  printf("gcd(%d, %d) = %d\n", a, b, GCD(a, b));

  return 0;
}

// gcd(130, 124) = 2
// gcd(662, 414) = 2
int GCD(int a, int b) {
  if (b == 0) {
    return a;
  }

  return GCD(b, a % b);
}
```

```c
/**
 * file: merge-sort.c
 *
 * Created by hengxin on 11/14/21.
 */

#include <stdio.h>

//#define LEN 7
//int numbers[LEN] = {38, 27, 43, 3, 9, 82, 10};

#define LEN 10
int numbers[LEN] = {4, 2, 8, 6, 0, 5, 1, 7, 3, 9};

void MergeSort(int nums[], int left, int right);

/**
 * Merge two subarrays nums[left .. mid] and nums[mid + 1 .. right]
 *
 * @param nums
 * @param left
 * @param mid
 * @param right
 */
void Merge(int nums[], int left, int mid, int right);

int main() {
  MergeSort(numbers, 0, LEN - 1);

  for (int i = 0; i < LEN; i++) {
    printf("%d ", numbers[i]);
  }

  return 0;
}

void MergeSort(int nums[], int left, int right) {
  if (left == right) {
    return;
  }

  int mid = (left + right) / 2;
  MergeSort(nums, left, mid);
  MergeSort(nums, mid + 1, right);

  Merge(nums, left, mid, right);
}

void Merge(int nums[], int left, int mid, int right) {
  /**
   * Create two temporary arrays
   * Using VLA (variable-length arrays)
   */
```

```
54   int left_size = mid - left + 1;
55   int nums_left[left_size];
56   for (int i = 0; i < left_size; i++) {
57     nums_left[i] = nums[left + i];
58   }
59
60   int right_size = right - mid;
61   int nums_right[right_size];
62   for (int i = 0; i < right_size; i++) {
63     nums_right[i] = nums[mid + 1 + i];
64   }
65
66   int left_index = 0;
67   int right_index = 0;
68   int current_index = left;
69
70   while (left_index < left_size && right_index < right_size) {
71     if (nums_left[left_index] <= nums_right[right_index]) {
72       nums[current_index] = nums_left[left_index];
73       left_index++;
74     } else {
75       nums[current_index] = nums_right[right_index];
76       right_index++;
77     }
78
79     current_index++;
80   }
81
82   while (left_index < left_size) {
83     nums[current_index] = nums_left[left_index];
84     left_index++;
85     current_index++;
86   }
87
88   while (right_index < right_size) {
89     nums[current_index] = nums_right[right_index];
90     right_index++;
91     current_index++;
92   }
93 }
```

```c
/**
 * file: min.c
 *
 * Recursively find the minimum of an array of integers
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>

#define NUM 5
int numbers[NUM] = {0};

int Min(const int nums[], int len);
int MinOfTwo(int a , int b);

int main() {
  for (int i = 0; i < NUM; i++) {
    scanf("%d", &numbers[i]);
  }

  printf("min = %d\n", Min(numbers, NUM));

  return 0;
}

int Min(const int nums[], int len) {
  if (len == 1) {
    return nums[0];
  }

  return MinOfTwo(nums[len - 1], Min(nums, len - 1));
}

int MinOfTwo(int a, int b) {
  return a > b ? b : a;
}
```

```c
/**
 * file: sum.c
 *
 * Recursively computing the sum of an array of integers
 *
 * Created by hengxin on 11/13/21.
 */

#include <stdio.h>

int Sum(const int numbers[], int len);

int main() {
  int numbers[] = {1, 2, 3, 4, 5};
    printf("sum = %d\n", Sum(numbers, sizeof numbers / sizeof numbers[0]));

  return 0;
}

int Sum(const int numbers[], int len) {
  if (len == 0) {
    return 0;
  }

  return numbers[len - 1] + Sum(numbers, len - 1);
}
```

```
 1 # 6-recursion
 2
 3 ## Recursion
 4 - `sum.c`
 5 - `min.c`
 6 - `fib.c`
 7 - `gcd.c`
 8 - `binary-search.c`
 9 - `merge-sort.c`
10
11 ## Backup
12 - `hanoi.c`
13 - `quicksort.c`
```