题目描述

游戏里面，队伍通过匹配实力相近的对手进行对战。但是如果匹配的队伍实力相差太大，对于双方游戏体验都不会太好。

给定n个队伍的实力值，对其进行两两实力匹配，两支队伍实例差距在允许的最大差距d内，则可以匹配。
要求在匹配队伍最多的情况下匹配出的各组实力差距的总和最小。

输入描述

第一行，n，d。队伍个数n。允许的最大实力差距d。

- 2<=n <=50
- 0<=d<=100

第二行，n个队伍的实力值空格分割。

- 0<=各队伍实力值<=100

输出描述

匹配后，各组对战的实力差值的总和。若没有队伍可以匹配，则输出-1。

用例

输入	6 30 81 87 47 59 81 18
输出	57
说明	18与47配对，实力差距29 59与81配对，实力差距22 81与87配对，实力差距6 总实力差距29+22+6=57

输入	6 20 81 87 47 59 81 18
输出	12
说明	最多能匹配成功4支队伍。 47与59配对，实力差距12， 81与81配对，实力差距0。 总实力差距12+0=12

输入	4 10 40 51 62 73
输出	-1
说明	实力差距都在10以上， 没有队伍可以匹配成功。

题目解析

本题要求两两组队，并且两个队伍的实力差值必须小于等于d才能组队。

假设现在有4队，实力值分别为 10，20，30，40，而d=20

因此10可以和20组队，也可以和30组队，但是不能和40组队。

那么这里，我们到底是让10和20组队，还是让10和30组队呢?

假设，10和20组队了，那么剩下的只能30和40组队

假设，10和30组队了，那么剩下的只能20和40组队

现在，题目要求：

> 要求在匹配队伍最多的情况下匹配出的各组实力差距的总和最小。

题目解析

本题要求两两组队，并且两个队伍的实力差值必须小于等于d才能组队。

假设现在有4队，实力值分别为 10，20，30，40，而d=20

因此10可以和20组队，也可以和30组队，但是不能和40组队。

那么这里，我们到底是让10和20组队，还是让10和30组队呢?

假设，10和20组队了，那么剩下的只能30和40组队

假设，10和30组队了，那么剩下的只能20和40组队

现在，题目要求：

> 要求在匹配队伍最多的情况下匹配出的各组实力差距的总和最小。

这里有两个条件，但是最优先的是要匹配队伍最多

那么上面例子中，两种组队方式，哪种能产生最多匹配队伍呢?

这里其实就是 贪心 思维，如果10和30组队，那么20和40组队的风险就增加了，因为30和40组队要比20和40组队的风险更低，即实力差值更小，更有可能组队成功。

因此，本题，我们可以先将队伍按照实力值从小到达排序，比如用例1：

> 18 47 59 81 81 87

然后，求出所有相邻队伍实力差值，并只保留实力差之小于等于d的组合，例如用例（升序后）可得如下组合：

- [0, 1, 29]　// 含义是：队伍0（18实力）和队伍1（47实力）组合，实力差为29
- [1, 2, 12]
- [2, 3, 22]
- [3, 4, 0]
- [4, 5, 6]

然后开始进行组合之间的组合，而组合之间进行组合的条件，即不存在重复队伍，比如

[0, 1, 29] 和 [1, 2, 12] 不能进行组合，因为有重复队伍1。

最终我们会得到多个多组合，此时只要取 组合数 最多的，如果存在组合数相同的多组合，则看实力差值最小的。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, d] = lines[0].split(" ").map(Number);
    const arr = lines[1].split(" ").map(Number);
    console.log(getResult(n, d, arr));
    lines.length = 0;
  }
});

function getResult(n, d, arr) {
  arr.sort((a, b) => a - b);

  const diffs = [];

  for (let i = 1; i < arr.length; i++) {
    const diff = arr[i] - arr[i - 1];
    if (diff <= d) {
      diffs.push([i - 1, i, diff]);
    }
  }

  if (diffs.length == 0) {
    return -1;
  }

  const res = [];
  dfs(0, diffs, [], res);

  res.sort((a, b) => (a[0] === b[0] ? a[1] - b[1] : b[0] - a[0]));

  return res[0][1];
}

```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, d] = lines[0].split(" ").map(Number);
    const arr = lines[1].split(" ").map(Number);
    console.log(getResult(n, d, arr));
    lines.length = 0;
  }
});

function getResult(n, d, arr) {
  arr.sort((a, b) => a - b);

  const diffs = [];

  for (let i = 1; i < arr.length; i++) {
    const diff = arr[i] - arr[i - 1];
    if (diff <= d) {
      diffs.push([i - 1, i, diff]);
    }
  }

  if (diffs.length == 0) {
    return -1;
  }

  const res = [];
  dfs(0, diffs, [], res);

  res.sort((a, b) => (a[0] === b[0] ? a[1] - b[1] : b[0] - a[0]));

  return res[0][1];
}

function dfs(index, diffs, path, res) {
  for (let i = index; i < diffs.length; i++) {
    if (path.length == 0 || path.at(-1)[1] < diffs[i][0]) {
      path.push(diffs[i]);
      dfs(i + 1, diffs, path, res);

      const count = path.length;
      const sumDiff = path.map((e) => e[2]).reduce((p, c) => p + c);

      res.push([count, sumDiff]);

      path.pop();
    }
  }
}
```

Java算法源码

```java
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int d = sc.nextInt();

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
    }

    System.out.println(getResult(n, d, arr));
  }

  public static int getResult(int n, int d, int[] arr) {
    Arrays.sort(arr);

    ArrayList<Integer[]> diffs = new ArrayList<>();

    for (int i = 1; i < arr.length; i++) {
      int diff = arr[i] - arr[i - 1];
      if (diff <= d) diffs.add(new Integer[] {i - 1, i, diff});
    }

    if (diffs.size() == 0) {
```

Java算法源码

```java
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int d = sc.nextInt();

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
    }

    System.out.println(getResult(n, d, arr));
  }

  public static int getResult(int n, int d, int[] arr) {
    Arrays.sort(arr);

    ArrayList<Integer[]> diffs = new ArrayList<>();

    for (int i = 1; i < arr.length; i++) {
      int diff = arr[i] - arr[i - 1];
      if (diff <= d) diffs.add(new Integer[] {i - 1, i, diff});
    }

    if (diffs.size() == 0) {
      return -1;
    }

    ArrayList<Integer[]> res = new ArrayList<>();
    dfs(0, diffs, new LinkedList<>(), res);

    res.sort((a, b) -> Objects.equals(a[0], b[0]) ? a[1] - b[1] : b[0] - a[0]);
    return res.get(0)[1];
  }

  public static void dfs(
      int index, ArrayList<Integer[]> diffs, LinkedList<Integer[]> path, ArrayList<Integer[]> res) {
    for (int i = index; i < diffs.size(); i++) {
      if (path.size() == 0 || path.getLast()[1] < diffs.get(i)[0]) {
        path.add(diffs.get(i));
        dfs(i + 1, diffs, path, res);

        int count = path.size();
        int sumDiff = path.stream().map(e -> e[2]).reduce((p, c) -> p + c).orElse(0);

        res.add(new Integer[] {count, sumDiff});

        path.removeLast();
      }
    }
  }
}
```

Python算法源码

```python
import sys

# 输入获取
n, d = map(int, input().split())
arr = list(map(int, input().split()))


# 算法入口
def getResult(n, d, arr):
    arr.sort()

    diffs = []
    for i in range(1, len(arr)):
        diff = arr[i] - arr[i - 1]
        if diff <= d:
            diffs.append([i - 1, i, diff])

    if len(diffs) == 0:
        return -1

    res = []
    dfs(0, diffs, [], res)

    res.sort(key=lambda x: (-x[0], x[1]))

    return res[0][1]


def dfs(index, diffs, path, res):
    for i in range(index, len(diffs)):
        if len(path) == 0 or path[-1][1] < diffs[i][0]:
            path.append(diffs[i])
```

Python算法源码

复制

```python
import sys

# 输入获取
n, d = map(int, input().split())
arr = list(map(int, input().split()))


# 算法入口
def getResult(n, d, arr):
    arr.sort()

    diffs = []
    for i in range(1, len(arr)):
        diff = arr[i] - arr[i - 1]
        if diff <= d:
            diffs.append([i - 1, i, diff])

    if len(diffs) == 0:
        return -1

    res = []
    dfs(0, diffs, [], res)

    res.sort(key=lambda x: (-x[0], x[1]))

    return res[0][1]


def dfs(index, diffs, path, res):
    for i in range(index, len(diffs)):
        if len(path) == 0 or path[-1][1] < diffs[i][0]:
            path.append(diffs[i])
            dfs(i + 1, diffs, path, res)

            count = len(path)
            sumDiff = sum(map(lambda x: x[2], path))

            res.append([count, sumDiff])
            path.pop()


# 算法调用
print(getResult(n, d, arr))
```