# Problem 3627: Maximum Median Sum of Subsequences of Size 3

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given an integer array

nums

with a length divisible by 3.

You want to make the array empty in steps. In each step, you can select any three elements from the array, compute their

median

, and remove the selected elements from the array.

The

median

of an odd-length sequence is defined as the middle element of the sequence when it is sorted in non-decreasing order.

Return the

maximum

possible sum of the medians computed from the selected elements.

Example 1:

Input:

nums = [2,1,3,2,1,3]

Output:

5

Explanation:

In the first step, select elements at indices 2, 4, and 5, which have a median 3. After removing these elements,

nums

becomes

[2, 1, 2]

.

In the second step, select elements at indices 0, 1, and 2, which have a median 2. After removing these elements,

nums

becomes empty.

Hence, the sum of the medians is

3 + 2 = 5

.

Example 2:

Input:

nums = [1,1,10,10,10,10]

Output:

20

Explanation:

In the first step, select elements at indices 0, 2, and 3, which have a median 10. After removing these elements,

nums

becomes

[1, 10, 10]

.

In the second step, select elements at indices 0, 1, and 2, which have a median 10. After removing these elements,

nums

becomes empty.

Hence, the sum of the medians is

10 + 10 = 20

.

Constraints:

1 <= nums.length <= 5 * 10

5

nums.length % 3 == 0

1 <= nums[i] <= 10

9

## Code Snippets

**C++:**

```
class Solution {
public:
long long maximumMedianSum(vector<int>& nums) {

}
};
```

**Java:**

```
class Solution {
public long maximumMedianSum(int[] nums) {

}
}
```

**Python3:**

```
class Solution:
def maximumMedianSum(self, nums: List[int]) -> int:
```

**Python:**

```
class Solution(object):
def maximumMedianSum(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} nums
* @return {number}
*/
var maximumMedianSum = function(nums) {

};
```

**TypeScript:**

```
function maximumMedianSum(nums: number[]): number {

};
```

**C#:**

```
public class Solution {
public long MaximumMedianSum(int[] nums) {

}
}
```

**C:**

```
long long maximumMedianSum(int* nums, int numsSize) {

}
```

**Go:**

```
func maximumMedianSum(nums []int) int64 {

}
```

**Kotlin:**

```
class Solution {
fun maximumMedianSum(nums: IntArray): Long {

}
}
```

**Swift:**

```
class Solution {
func maximumMedianSum(_ nums: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn maximum_median_sum(nums: Vec<i32>) -> i64 {

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @return {Integer}
def maximum_median_sum(nums)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maximumMedianSum($nums) {

}
}
```

**Dart:**

```
class Solution {
int maximumMedianSum(List<int> nums) {

}
}
```

### Scala:

```
object Solution {
def maximumMedianSum(nums: Array[Int]): Long = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec maximum_median_sum(nums :: [integer]) :: integer
def maximum_median_sum(nums) do

end
end
```

### Erlang:

```
-spec maximum_median_sum(Nums :: [integer()]) -> integer().
maximum_median_sum(Nums) ->
.
```

### Racket:

```
(define/contract (maximum-median-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
long long maximumMedianSum(vector<int>& nums) {

}
};
```

## Java Solution:

```
/**
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long maximumMedianSum(int[] nums) {

}
}
```

## Python3 Solution:

```
"""
Problem: Maximum Median Sum of Subsequences of Size 3
Difficulty: Medium
Tags: array, greedy, math, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maximumMedianSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

**Python Solution:**

```
class Solution(object):
def maximumMedianSum(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

**JavaScript Solution:**

```
/**
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @return {number}
*/
var maximumMedianSum = function(nums) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function maximumMedianSum(nums: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long MaximumMedianSum(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum Median Sum of Subsequences of Size 3
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

long long maximumMedianSum(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Maximum Median Sum of Subsequences of Size 3
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maximumMedianSum(nums []int) int64 {

}
```

**Kotlin Solution:**

```
class Solution {
fun maximumMedianSum(nums: IntArray): Long {

}
}
```

**Swift Solution:**

```
class Solution {
func maximumMedianSum(_ nums: [Int]) -> Int {

}
}
```

**Rust Solution:**

```
// Problem: Maximum Median Sum of Subsequences of Size 3
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximum_median_sum(nums: Vec<i32>) -> i64 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[]} nums
# @return {Integer}
def maximum_median_sum(nums)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maximumMedianSum($nums) {

}
}
```

**Dart Solution:**

```
class Solution {
int maximumMedianSum(List<int> nums) {

}
}
```

**Scala Solution:**

```
object Solution {
def maximumMedianSum(nums: Array[Int]): Long = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec maximum_median_sum(nums :: [integer]) :: integer
def maximum_median_sum(nums) do

end
end
```

## Erlang Solution:

```
-spec maximum_median_sum(Nums :: [integer()]) -> integer().
maximum_median_sum(Nums) ->
.
```

## Racket Solution:

```
(define/contract (maximum-median-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```