Conditions: a non-decreasing order

My solution:

1. Check zero: put zero in the far left.
2. If non-zero:
   1. Easy: all members are positive numbers or negative numbers
      1. Positive: directly perform the square operation
      2. Negative: perform the square operation and reversal store
   2. Different: positive and negative
      1. Change negative numbers into positive
      2. Check non-decreasing order
         1. True: like easy positive
         2. False: like easy negative

第一步：找到正数与负数的交接点i.

第二部：由交接点开始不断左右比较大小。小的数进行平方放入待存储的容器中。

我的解决方案以及运行效率

|  |
| --- |
| class Solution {  private:  //bool zero = 0;  //bool positive = 0;  //bool dec\_order = 0;    int check\_posi(const vector<int>& A) {//检查正数的位置  for (int i = 0; i < A.size(); i++) {  if(A[i] >= 0) {  return i;  }  }  return A.size();  }  int abs(const int num) {//平方函数  if(num >= 0)  return num;  return -num;  }  vector<int> perform\_order(const vector<int>& A,int k) {//从k出开始进行比较  vector<int> B;  int key1 = k - 1, key2 = k;  for(int i = 0;i < A.size();i++) {  if(key1 >= 0 && key2 < A.size()) {  if(abs(A[key1]) >= A[key2]) {  B.push\_back(A[key2]\*A[key2]);  ++key2;  }  else {  B.push\_back(A[key1]\*A[key1]);  --key1;  }  }  else if(key1 < 0 && key2 < A.size()) {  B.push\_back(A[key2]\*A[key2]);  ++key2;  }  else if(key1 >= 0 && key2 >= A.size()) {  B.push\_back(A[key1]\*A[key1]);  --key1;  }  }  return B;  }  public:  vector<int> sortedSquares(const vector<int>& A) {  //vector<int> B(A);  int i = check\_posi(A);  //square(B);  return perform\_order(A,i);  }  }; |
| Runtime: 112 ms, faster than 99.21% of C++ online submissions for Squares of a Sorted Array.  Memory Usage: 15.1 MB, less than 100.00% of C++ online submissions for Squares of a Sorted Array. |

同种思路，但是效率更高的解决方案：

C++：

|  |
| --- |
| vector<int> sortedSquares(vector<int>& A) {  vector<int> result(A.size());  ssize\_t l = 0;  ssize\_t r = A.size();  ssize\_t p = A.size() - 1;    while (l < r) {  int x = A[l] \* A[l];  int y = A[r-1] \* A[r-1];    if (x >= y) {  result[p] = x;  p--;  l++;  } else {  result[p] = y;  p--;  r--;  }  }    return result;  } |
| vector<int> sortedSquares(vector<int>& A) {  int len = A.size();  vector<int> res(len, 0);  int i = 0,j = len - 1;  for(int k = len-1; k >= 0; k--){  if(abs(A[i]) > abs(A[j]))  res[k] = pow(A[i++], 2);  else  res[k] = pow(A[j--], 2);  }  return res;  } |

Java:

|  |
| --- |
| public int[] sortedSquares(int[] A) {  int[] r = new int[A.length];  for(int d=A.length-1,u=0,i=d; i>=0; i--) r[i] = -A[u]>A[d] ? A[u]\*A[u++] : A[d]\*A[d--];  return r;  } |