kwikSort_iterative

v1.0

Generated on Tue Oct 28 2025 21:11:16 for kwikSort_iterative by Doxygen 1.9.8

Tue Oct 28 2025 21:11:16

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

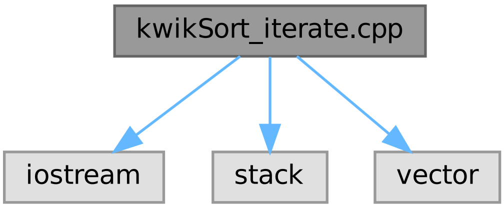
File Documentation

2.1 kwikSort_iterate.cpp File Reference

Implementation of a non-recursive (iterative) quicksort algorithm using a stack.

```
#include <iostream>
#include <stack>
#include <vector>
```

Include dependency graph for kwikSort_iterate.cpp:



Functions

- int main ()
- int partition (vector< int > &set, int start, int end)

  Partitions a subarray around a pivot for quicksort.
- void quickSort (vector< int > &set, int start, int end)

  Performs an iterative (non-recursive) quicksort on a vector.

2.1.1 Detailed Description

Implementation of a non-recursive (iterative) quicksort algorithm using a stack.

This program demonstrates an iterative version of the quicksort algorithm. It replaces recursion with an explicit stack to manage subarray partitions.

Author:

- David J. Devney

Date

2025-02-15 Version:
1.0 @course CSCI 331

See also:
partition() See also:
quickSort()

Warning:
Ensure sufficient stack capacity for large data sets. Known bug:
None currently known. To-do:
Extend to support descending order sorting or custom comparators.

Definition in file kwikSort_iterate.cpp.

2.1.2 Function Documentation

2.1.2.1 main()

```
int main ( )
```

Examples

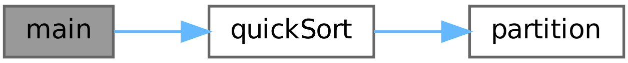
/workspaces/HW_3_331/kwikSort_iterate.cpp.

Definition at line 137 of file kwikSort_iterate.cpp.

```
00138 {
00139     vector<int> set = {10, 7, 8, 9, 1, 5};
00140 
00141     cout « "Original array: ";
00142     for (int num : set)
00143     {
00144         cout « num « " ";
00145     }
00146     cout « endl;
00147 
00148     quickSort(set, 0, set.size() - 1);
00149 
00150     cout « "Sorted array: ";
00151     for (int num : set)
00152     {
00153         cout « num « " ";
00154     }
00155     cout « endl;
00156 
00157     return 0;
00158 }
```

References quickSort().

Here is the call graph for this function:



2.1.2.2 partition()

```
int partition (
              vector< int > & set,
              int start,
              int end )
```

Partitions a subarray around a pivot for quicksort.

The partition function selects a pivot value and reorders the subarray such that all elements less than the pivot are moved before it, and all elements greater than or equal to the pivot are moved after it.

Parameters

in,out	*set*	The vector of integers to partition.
in	*start*	The starting index of the subarray to partition.
in	*end*	The ending index of the subarray to partition.

Precondition:
start and end must be valid indices within the bounds of set. Postcondition:
Elements in set are rearranged such that all elements before the pivot are smaller and all elements after are greater or equal.

Returns

The index position of the pivot after partitioning.

See also:
quickSort()

Note

Uses median-of-three pivot selection for improved performance.

Examples

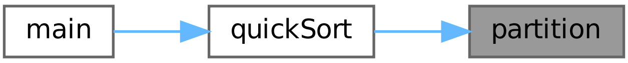
/workspaces/HW_3_331/kwikSort_iterate.cpp.

Definition at line 52 of file kwikSort_iterate.cpp.

```
00053 {
00054     int pivotValue, pivotIndex, mid;
00055 
00056     mid = (start + end) / 2;
00057     swap(set[start], set[mid]);
00058     pivotIndex = start;
00059     pivotValue = set[start];
00060 
00061     for (int scan = start + 1; scan <= end; scan++)
00062     {
00063         if (set[scan] < pivotValue)
00064         {
00065             pivotIndex++;
00066             swap(set[pivotIndex], set[scan]);
00067         }
00068     }
00069     swap(set[start], set[pivotIndex]);
00070     return pivotIndex;
00071 }
```

Referenced by quickSort().

Here is the caller graph for this function:



2.1.2.3 quickSort()

```
void quickSort (
            vector< int > & set,
            int start,
            int end )
```

Performs an iterative (non-recursive) quicksort on a vector.

This function sorts a vector of integers in ascending order using the quicksort algorithm implemented with an explicit stack instead of recursive function calls.

Parameters

`in,out`	*set*	The vector of integers to be sorted.
`in`	*start*	The starting index of the vector (typically 0).
`in`	*end*	The ending index of the vector (typically `set.size() - 1`).

Precondition:
`start` and `end` must be valid indices in `set`, with `start <= end`. Postcondition:
The vector `set` will be sorted in ascending order.

Returns

void

See also:
partition() Test case:
Example:

```
vector<int> nums = {10, 7, 8, 9, 1, 5};
quickSort(nums, 0, nums.size() - 1);
// nums is now {1, 5, 7, 8, 9, 10}
```

Examples

/workspaces/HW_3_331/kwikSort_iterate.cpp.

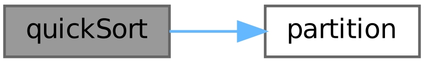
Definition at line 99 of file kwikSort_iterate.cpp.

```
{
    stack<pair<int, int» s;
    s.push({start, end});

    while (!s.empty())
    {
        int start = s.top().first;
        int end = s.top().second;
        s.pop();

        if (start < end)
        {
            int p = partition(set, start, end);

            s.push({start, p - 1});
            s.push({p + 1, end});
        }
    }
}
```

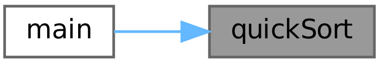
References partition().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



2.2 kwikSort_iterate.cpp

Go to the documentation of this file.

```
/**
 * @file kwikSort_iterate.cpp
 * @brief Implementation of a non-recursive (iterative) quicksort algorithm using a stack.
 *
 * @details
 * This program demonstrates an iterative version of the quicksort algorithm.
 * It replaces recursion with an explicit stack to manage subarray partitions.
 *
 * @author
 * - David J. Devney
 *
 * @date 2025-02-15
 * @version 1.0
 * @course CSCI 331
 *
 * @see partition()
 * @see quickSort()
 *
 * @warning Ensure sufficient stack capacity for large data sets.
 * @bug None currently known.
 * @todo Extend to support descending order sorting or custom comparators.
 */

#include <iostream>
#include <stack>
#include <vector>

using namespace std;

/**
 * @brief Partitions a subarray around a pivot for quicksort.
 *
 * @details
 * The partition function selects a pivot value and reorders the subarray
 * such that all elements less than the pivot are moved before it, and all
 * elements greater than or equal to the pivot are moved after it.
 *
 * @param[in,out] set The vector of integers to partition.
 * @param[in] start The starting index of the subarray to partition.
 * @param[in] end The ending index of the subarray to partition.
 *
 * @pre `start` and `end` must be valid indices within the bounds of `set`.
 * @post Elements in `set` are rearranged such that all elements before
 *       the pivot are smaller and all elements after are greater or equal.
 *
 * @return The index position of the pivot after partitioning.
 *
 *
 * @see quickSort()
 * @note Uses median-of-three pivot selection for improved performance.
 */
int partition(vector<int>& set, int start, int end)
{
    int pivotValue, pivotIndex, mid;

    mid = (start + end) / 2;
    swap(set[start], set[mid]);
    pivotIndex = start;
    pivotValue = set[start];

    for (int scan = start + 1; scan <= end; scan++)
    {
        if (set[scan] < pivotValue)
        {
            pivotIndex++;
            swap(set[pivotIndex], set[scan]);
        }
    }
    swap(set[start], set[pivotIndex]);
    return pivotIndex;
}

/**
 * @brief Performs an iterative (non-recursive) quicksort on a vector.
 *
 * @details
 * This function sorts a vector of integers in ascending order using
 * the quicksort algorithm implemented with an explicit stack instead
 * of recursive function calls.
 *
 * @param[in,out] set The vector of integers to be sorted.
 * @param[in] start The starting index of the vector (typically 0).
 * @param[in] end The ending index of the vector (typically `set.size() - 1‘).
 *
 * @pre `start‘ and `end‘ must be valid indices in `set‘, with `start <= end‘.
 * @post The vector `set‘ will be sorted in ascending order.
 *
 * @return void
 *
 *
 * @see partition()
 * @test Example:
 * @code
 * vector<int> nums = {10, 7, 8, 9, 1, 5};
 * quickSort(nums, 0, nums.size() - 1);
 * // nums is now {1, 5, 7, 8, 9, 10}
 * @endcode
 */
void quickSort(vector<int>& set, int start, int end)
{
    stack<pair<int, int» s;
    s.push({start, end});

    while (!s.empty())
    {
        int start = s.top().first;
        int end = s.top().second;
        s.pop();

        if (start < end)
        {
            int p = partition(set, start, end);

            s.push({start, p - 1});
            s.push({p + 1, end});
        }
    }
}

/**
 * @brief Entry point of the program.
 *
 * @details
 * Demonstrates the iterative quicksort algorithm by sorting a small
 * example vector and printing the results before and after sorting.
 *
 * @return Returns 0 upon successful completion.
 *
 * @see quickSort()
 * @example
 * Input:
 * @code
 * Original array: 10 7 8 9 1 5
 * Sorted array:   1 5 7 8 9 10
 * @endcode
 */
int main()
{
    vector<int> set = {10, 7, 8, 9, 1, 5};

    cout « "Original array: ";
    for (int num : set)
    {
        cout « num « " ";
    }
    cout « endl;

    quickSort(set, 0, set.size() - 1);

    cout « "Sorted array: ";
    for (int num : set)
    {
        cout « num « " ";
    }
    cout « endl;

    return 0;
}
```

Chapter 3

Examples

3.1 /workspaces/HW_3_331/kwikSort_iterate.cpp

Entry point of the program.

Entry point of the program.Demonstrates the iterative quicksort algorithm by sorting a small example vector and printing the results before and after sorting.

Returns

Returns 0 upon successful completion.

See also:
quickSort()

Input:

```
Original array: 10 7 8 9 1 5
Sorted array:   1 5 7 8 9 10
/**
 * @file kwikSort_iterate.cpp
 * @brief Implementation of a non-recursive (iterative) quicksort algorithm using a stack.
 *
 * @details
 * This program demonstrates an iterative version of the quicksort algorithm.
 * It replaces recursion with an explicit stack to manage subarray partitions.
 *
 * @author
 * - David J. Devney
 *
 * @date 2025-02-15
 * @version 1.0
 * @course CSCI 331
 *
 * @see partition()
 * @see quickSort()
 *
 * @warning Ensure sufficient stack capacity for large data sets.
 * @bug None currently known.
 * @todo Extend to support descending order sorting or custom comparators.
 */

#include <iostream>
#include <stack>
#include <vector>

using namespace std;

/**
 * @brief Partitions a subarray around a pivot for quicksort.
 *
 * @details
 * The partition function selects a pivot value and reorders the subarray
 * such that all elements less than the pivot are moved before it, and all
 * elements greater than or equal to the pivot are moved after it.
 *
 * @param[in,out] set The vector of integers to partition.
 * @param[in] start The starting index of the subarray to partition.
 * @param[in] end The ending index of the subarray to partition.
 *
 * @pre `start‘ and `end‘ must be valid indices within the bounds of `set‘.
 * @post Elements in `set‘ are rearranged such that all elements before
 *       the pivot are smaller and all elements after are greater or equal.
 *
 * @return The index position of the pivot after partitioning.
 *
 *
 * @see quickSort()
 * @note Uses median-of-three pivot selection for improved performance.
 */
int partition(vector<int>& set, int start, int end)
{
    int pivotValue, pivotIndex, mid;

    mid = (start + end) / 2;
    swap(set[start], set[mid]);
    pivotIndex = start;
    pivotValue = set[start];

    for (int scan = start + 1; scan <= end; scan++)
    {
        if (set[scan] < pivotValue)
        {
            pivotIndex++;
            swap(set[pivotIndex], set[scan]);
        }
    }
    swap(set[start], set[pivotIndex]);
    return pivotIndex;
}

/**
 * @brief Performs an iterative (non-recursive) quicksort on a vector.
 *
 * @details
 * This function sorts a vector of integers in ascending order using
 * the quicksort algorithm implemented with an explicit stack instead
 * of recursive function calls.
 *
 * @param[in,out] set The vector of integers to be sorted.
 * @param[in] start The starting index of the vector (typically 0).
 * @param[in] end The ending index of the vector (typically `set.size() - 1‘).
 *
 * @pre `start‘ and `end‘ must be valid indices in `set‘, with `start <= end‘.
 * @post The vector `set‘ will be sorted in ascending order.
 *
 * @return void
 *
 *
 * @see partition()
 * @test Example:
 * @code
 * vector<int> nums = {10, 7, 8, 9, 1, 5};
 * quickSort(nums, 0, nums.size() - 1);
 * // nums is now {1, 5, 7, 8, 9, 10}
 * @endcode
 */
void quickSort(vector<int>& set, int start, int end)
{
    stack<pair<int, int» s;
    s.push({start, end});

    while (!s.empty())
    {
        int start = s.top().first;
        int end = s.top().second;
        s.pop();

        if (start < end)
        {
            int p = partition(set, start, end);

            s.push({start, p - 1});
            s.push({p + 1, end});
        }
    }
}

/**
 * @brief Entry point of the program.
 *
 * @details
 * Demonstrates the iterative quicksort algorithm by sorting a small
 * example vector and printing the results before and after sorting.
 *
 * @return Returns 0 upon successful completion.
 *
 * @see quickSort()
 * @example
 * Input:
 * @code
 * Original array: 10 7 8 9 1 5
 * Sorted array:   1 5 7 8 9 10
 * @endcode
 */
int main()
{
    vector<int> set = {10, 7, 8, 9, 1, 5};

    cout « "Original array: ";
    for (int num : set)
    {
        cout « num « " ";
    }
    cout « endl;

    quickSort(set, 0, set.size() - 1);

    cout « "Sorted array: ";
    for (int num : set)
    {
        cout « num « " ";
    }
    cout « endl;

    return 0;
}
```

Index