CSCI331 Project 2

Joshua Ackerman, Erik Drake, Alex Kayser, Syed Shah Newaj, Steven Scholz, Ghislain Twagirayezu

Generated by Doxygen 1.8.12

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 TournamentSort< T >::compare Struct Reference

`#include <TournamentSort.h>`

Public Member Functions

- bool operator() (const Node &lhs, const Node &rhs)

3.1.1 Detailed Description

template<class T>
struct TournamentSort< T >::compare

Sorts the values

3.1.2 Member Function Documentation

3.1.2.1 operator()()

```
template<class T >
bool TournamentSort< T >::compare::operator() (
            const Node & lhs,
            const Node & rhs )  [inline]
```

The documentation for this struct was generated from the following file:

- TournamentSort.h

3.2 TournamentSort< T >::Node Class Reference

`#include <TournamentSort.h>`

Public Member Functions

- Node (int x, int y, T z)
- virtual ~Node ()

Public Attributes

- int position
- int rn
- T value

3.2.1 Detailed Description

template<class T>
class TournamentSort< T >::Node

This Node is to help preserve meta data of the value

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

```
template<class T >
TournamentSort< T >::Node::Node (
            int x,
            int y,
            T z )  [inline]
```

3.2.2.2 ~Node()

```
template<class T >
virtual TournamentSort< T >::Node::~Node ( )  [inline], [virtual]
```

3.2.3 Member Data Documentation

3.2.3.1 position

```
template<class T >
int TournamentSort< T >::Node::position
```

What position the value is in its respective list

3.2.3.2 rn

```
template<class T >
int TournamentSort< T >::Node::rn
```

Relative record number

3.2.3.3 value

```
template<class T >
T TournamentSort< T >::Node::value
```

The raw data that is being sorted

The documentation for this class was generated from the following file:

- TournamentSort.h

3.3 ReplacementSelectionSort< T > Class Template Reference

```
#include <ReplacementSelectionSort.h>
```

Public Member Functions

- ReplacementSelectionSort (const int _size, istream &_infile, ostream &_outputfile, const bool _sort)
- virtual ~ReplacementSelectionSort ()
- void sort (istream &infile, ostream &outputfile)
- void heapify (bool leftSide)
- void siftUp (int index, bool leftSide)
- void siftDown (int index, bool leftSide)
- int left (int index, bool leftSide)
- int right (int index, bool leftSide)
- int parent (int index, bool leftSide)
- void swap (int index1, int index2)
- T pop ()
- T initHeap (istream &infile)
- void current_dualHeap_push (const T entry)
- void pending_dualHeap_push (const T entry)
- void swapActive ()
- void sortPreMerge (istream &infile, ostream &outputfile)

Public Attributes

- bool activeLeftHeap
- bool direction_flag
- int size
- int leftHeapStart
- int leftHeapEnd
- int rightHeapStart
- int rightHeapEnd
- vector< T > dualHeap

3.3.1 Constructor & Destructor Documentation

3.3.1.1 ReplacementSelectionSort()

```
template<typename T >
ReplacementSelectionSort< T >::ReplacementSelectionSort (
            const int _size,
            istream & _infile,
            ostream & _outputfile,
            const bool _sort )
```

Construcor

Parameters

_size	represents the fixed memory size
_infile	input file that is to be sorted
_outputfile	output file after the process is complete
_sort	direction of the sorting True = ascending

3.3.1.2 ~ReplacementSelectionSort()

```
template<typename T >
ReplacementSelectionSort< T >::~ReplacementSelectionSort ( )  [virtual]
```

Default destructor

3.3.2 Member Function Documentation

3.3.2.1 current_dualHeap_push()

```
template<typename T >
void ReplacementSelectionSort< T >::current_dualHeap_push (
            const T entry )
```

Pushes an element onto the active/current side of the dual heap

Parameters

entry	Item to be pushed onto the active side of the dual heap

3.3.2.2 heapify()

```
template<typename T >
void ReplacementSelectionSort< T >::heapify (
            bool leftSide )
```

Builds the heap structure

Parameters

leftSide	which side of the dual heap to be heapified

3.3.2.3 initHeap()

```
template<typename T >
T ReplacementSelectionSort< T >::initHeap (
            istream & infile )
```

Initializes the heap

Parameters

infile	Input file to grab records from

Returns

T Returns the element grabed from the file

3.3.2.4 left()

```
template<typename T >
int ReplacementSelectionSort< T >::left (
            int index,
            bool leftSide )
```

Return left indice of a given element

Parameters

index	Index of given element
leftSide	which side of the dual heap to look at

Returns

Return left indice of a given element

3.3.2.5 parent()

```
template<typename T >
int ReplacementSelectionSort< T >::parent (
            int index,
            bool leftSide )
```

Return parent indicie of a given element

Parameters

index	Index of a given element
leftSide	which side of the dual heap to loo at

Returns

Return parent indicie of a given element

3.3.2.6 pending_dualHeap_push()

```
template<typename T >
```

```
void ReplacementSelectionSort< T >::pending_dualHeap_push (
            const T entry )
```

Pushes an element onto the pending side of the dual heap

Parameters

entry	Item to be pushed onto the pending side of the dual heap

3.3.2.7 pop()

```
template<typename T >
T ReplacementSelectionSort< T >::pop ( )
```

Pops the top element in the heap

Returns

Returns the value of the poped element

3.3.2.8 right()

```
template<typename T >
int ReplacementSelectionSort< T >::right (
            int index,
            bool leftSide )
```

Return right indice of a given alement

Parameters

index	Index of a given element
leftSide	which side of the dual heap to look at

Returns

Return right indice of a given element

3.3.2.9 siftDown()

```
template<typename T >
void ReplacementSelectionSort< T >::siftDown (
            int index,
            bool leftSide )
```

Operation to help build the heap

Parameters

index	Index of the item to be sifted down
leftSide	which side of the dual heap to look at

3.3.2.10 siftUp()

```
template<typename T >
void ReplacementSelectionSort< T >::siftUp (
            int index,
            bool leftSide )
```

Operation to help build the heap

Parameters

index	Index of the item to be sifted up
leftSide	which side of the dual heap to look at

3.3.2.11 sort()

```
template<class T >
void ReplacementSelectionSort< T >::sort (
            istream & infile,
            ostream & outputfile )
```

Uses replacement selection sort to create multiple sorted lists

Parameters

infile	A file of records to be sorted
outputfile	A file to put the multiple sorted records

3.3.2.12 sortPreMerge()

```
template<class T >
void ReplacementSelectionSort< T >::sortPreMerge (
            istream & infile,
            ostream & outputfile )
```

Helper function for sorting using the replacement selection sort algorithm

Parameters

infile	Inputfile to be read
outputfile	Outputfile to be writen to

3.3.2.13 swap()

```
template<typename T >
void ReplacementSelectionSort< T >::swap (
            int index1,
            int index2 )
```

Swaps two elements

Parameters

index1	index of the first element to be swapped
index2	index of the second element to be swapped

3.3.2.14 swapActive()

```
template<typename T >
void ReplacementSelectionSort< T >::swapActive ( )
```

Swaps which active and pending sides

3.3.3 Member Data Documentation

3.3.3.1 activeLeftHeap

```
template<class T >
bool ReplacementSelectionSort< T >::activeLeftHeap
```

True = left heap is the active heap

3.3.3.2 direction_flag

```
template<class T >
bool ReplacementSelectionSort< T >::direction_flag
```

Sorting order True = ascending

3.3.3.3 dualHeap

```
template<class T >
vector<T> ReplacementSelectionSort< T >::dualHeap
```

A vector that will serve as the dual heap ADT

3.3.3.4 leftHeapEnd

```
template<class T >
int ReplacementSelectionSort< T >::leftHeapEnd
```

Position of the ending of the left heap

3.3.3.5 leftHeapStart

```
template<class T >
int ReplacementSelectionSort< T >::leftHeapStart
```

Position of the begining of the left heap

3.3.3.6 rightHeapEnd

```
template<class T >
int ReplacementSelectionSort< T >::rightHeapEnd
```

Position of the ending of the right heap

3.3.3.7 rightHeapStart

```
template<class T >
int ReplacementSelectionSort< T >::rightHeapStart
```

Position of the begining of the right heap

3.3.3.8 size

```
template<class T >
int ReplacementSelectionSort< T >::size
```

Fixed memory size

The documentation for this class was generated from the following files:

- ReplacementSelectionSort.h
- ReplacementSelectionSort.cpp

3.4 TournamentSort< T > Class Template Reference

```
#include <TournamentSort.h>
```

Classes

- struct compare
- class Node

Public Member Functions

- TournamentSort ()
- virtual ~TournamentSort ()
- std::priority_queue< Node, std::vector< Node >, compare > initPQ (std::priority_queue< Node, std::vector< Node >, compare > PQ, std::vector< vector< T > > sortedLists)
- void pushToFinal (std::priority_queue< Node, std::vector< Node >, compare > PQ, vector< T > final, std::vector< vector< T > > sortedLists)

Public Attributes

- int completedRuns
- vector< T > finalVector

3.4.1 Detailed Description

template<class T>
class TournamentSort< T >

Main TournamentSort class

3.4.2 Constructor & Destructor Documentation

3.4.2.1 TournamentSort()

```
template<class T >
TournamentSort< T >::TournamentSort ( )  [inline]
```

Default constructor

3.4.2.2 ~TournamentSort()

```
template<class T >
virtual TournamentSort< T >::~TournamentSort ( )  [inline], [virtual]
```

Default destructor

3.4.3 Member Function Documentation

3.4.3.1 initPQ()

```
template<class T >
std::priority_queue<Node, std::vector<Node>,compare> TournamentSort< T >::initPQ (
            std::priority_queue< Node, std::vector< Node >, compare > PQ,
            std::vector< vector< T > > sortedLists )  [inline]
```

Initializes the priority queue

Parameters

PQ	a priority queue of nodes containing values and other information
sortedLists	a vector of vectors containing sorted runs

Returns

priority_queue PQ

3.4.3.2 pushToFinal()

```
template<class T >
void TournamentSort< T >::pushToFinal (
            std::priority_queue< Node, std::vector< Node >, compare > PQ,
            vector< T > final,
            std::vector< vector< T > > sortedLists )  [inline]
```

Pushes the smallest (unless sorting is changed) item in the priority queue and repopulates it

Parameters

PQ	priority queue of nodes
final	a vector containing merged runs
sortedLists	a vector of vectors containing sorted runs

3.4.4 Member Data Documentation

3.4.4.1 completedRuns

```
template<class T >
int TournamentSort< T >::completedRuns
```

Number of completed runs

3.4.4.2 finalVector

```
template<class T >
vector<T> TournamentSort< T >::finalVector
```

The final vector containing a single sorted list

The documentation for this class was generated from the following file:

- TournamentSort.h

Chapter 4

File Documentation

4.1 ascending_randomDoubleTest_out.cpp File Reference

4.1.1 Detailed Description

84 83 79 56 45 44 43 31 30 28 24 24 20 17 14 14 13 10
93 80 68 65 55 52 49 46 41 41 36 32 32 23 23 22 21 18
96 95 86 71 67 63 62 60 56 53 52 48 47 42 34 30 28 17 13
98 95 91 85 78 73 66 65 61 59 48 47 38 34 27 26 23 14 13 13
98 87 76 73 65 63 60 57 50 46 45 41 38 35 32 32 22 21 10
74 71 66 66 66 59 51 48 48 38 32 30 14
91 90 89 78 77 77 55 52 48 46 44 42 40 31 28 19 18 14 12
93 86 84 69 67 67 60 49 46 36 31 28 27 24 16 11
95 92 83 76 74 66 57 56 46 44 38 37 32 31 18 17 12
87 83 82 80 73 62 57 55 55 55 46 45 41 41 31 29 28 28 18 18 10
97 92 85 84 77 74 73 71 55 50 48 33 31 31 21 19 17 16 16 12
98 81 79 75 73 64 54 50 50 47 40 31 17
96 88
averageRunLength = 16
the file is sorted: True
Max run length: 21
Min run length: 2
total Runs: 13
total items: 215

4.2 ascending_randomFloatTest_out.cpp File Reference

4.2.1 Detailed Description

93 88 86 64 62 58 48 40 34 23 18 15
95 90 87 86 85 83 66 52 49 48 44 40 38 33 33 28 21 18 14 10
88 73 70 69 63 61 59 57 56 53 52 51 28 24 24 23 12
97 91 86 84 82 80 77 74 68 59 59 49 39 34 33 27 25 21 17 12
94 92 83 82 82 78 71 67 58 53 51 49 48 44 33 21 18 14
95 94 91 80 78 77 77 72 72 67 65 63 61 53 43 42 36 32 32 20 19

93 92 85 78 77 76 68 56 50 47 43 43 30 25 22 21 16 14 13 11
86 77 76 72 69 48 48 42 36 31 23 17 15
94 94 89 87 76 75 75 66 66 50 41 34 25 21 16 12
98 95 94 89 83 76 69 60 59 59 55 51 51 46 32 29 27 20 16 14
98 95 94 92 85 85 80 77 74 74 67 66 64 63 62 51 45 37 32 29 28 20 10
95 93 88 87 87 87 72 68 66 64 57 57 55 47 46 45 36 31 25 24 14
92 91 82 79 75 70 65 62 62 53 49 47 47 44 43 43 35 26 26 15
90 88 87 73 63 57 55 54 53 53 43 41 40 38 29 26 25 24 19 18 13
97 93 80 79 77 75 59 54 52 51 49 35 35 33 31 28 28 23 16 12
95 91 77 76 55 54 43
averageRunLength = 18
the file is sorted: True
Max run length: 23
Min run length: 7
total Runs: 16
total items: 289

4.3 ascending_randomIntTest_out.cpp File Reference

4.3.1 Detailed Description

79 75 62 61 54 51 48 46 44 32 29 19 17 13 11
98 97 81 80 79 76 76 63 62 53 50 46 42 37 35 35 29 27 25 25
95 93 91 87 76 74 74 72 69 66 64 56 37 33 32 28 25 24 16 14 10 10
95 87 82 78 75 72 68 67 62 58 53 48 47 42 28 26 25 22
89 88 83 81 80 79 72 71 67 61 60 56 54 49 45 33 29 13 13
98 90 88 87 87 85 82 75 72 61 55 48 36 33 25 19 15 15 11
98 91 89 89 85 85 77 75 71 70 67 45 44 37 36 36 35 33 32 26
93 92 87 85 84 74 70 59 57 53 49 47 47 46 44 43 39 27 27 26 13 10
93 87 84 75 69 67 65 61 60 59 58 55 52 27
94 80 80 79
averageRunLength = 17
the file is sorted: True
Max run length: 22
Min run length: 4
total Runs: 10
total items: 173

4.4 ascending_randomStringTest_out.cpp File Reference

4.4.1 Detailed Description

ji jE dy dT cY bW Yq Wi JW JL JI FD Eb Da DW CY CE Bh AB
zv xx tX kp iq gn eS eG dK QR Pi Pa NM MM Hu Gs EA Ar AZ
wv ra ot nF mj lS cB bv aU Vy Vt VZ Pf MT IM HD Gh Gb
zi vR tq sK rb
averageRunLength = 15
the file is sorted: True
Max run length: 19
Min run length: 5
total Runs: 4
total items: 61

4.5 descending_randomDoubleTest_out.cpp File Reference

4.5.1 Detailed Description

90 82 80 76 76 66 62 57 57 47 32 21 19 18 14
97 97 90 90 84 82 77 69 65 62 59 49 48 47 41 39 33 32 25 20 18
95 91 82 81 80 79 77 70 62 60 59 57 54 53 46 38 36 34 28 20 19 17 15 13
97 97 97 92 87 85 77 73 68 66 63 52 52 47 44 41 37 28
95 92 90 89 87 84 76 76 74 66 65 63 63 57 51 48 42 33 31 18 12 10 10
92 91 90 89 81 78 72 69 68 66 65 55 33 33 31 31 29 19 18 15 12
98 93 93 86 85 82 78 77 73 71 63 61 50 39 36 31 30 27 14 13
96 92 92 86 82 79 77 68 67 59 59 46 45 39 39 34 28 26 26 25 25 15
93 91 83 80 67 56 53 45 38 36 33 31 28 27 25 10
94 93 92 89 77 76 75 58 57 56 51 51 37 35 21 20 19 18
98 94 90 86 83 80 70 69 69 68 66 62 61 57 54 52 49 48 47 43 43 35 28 27 25 20 15 14
98 93 85 83 75 64 63 56 53 51 41 35 35 33 32 31 29 17
96 92 89 81 81 78 76 61 59 58 56 53 49 47 41 38 35 24 24 13 12
98 97 96 81 79 77 72 67 66 62 53 52 50 45 31 30 18 18 17 15
97 93 93 87 86 82 77 66 62 51 48 40 39 33 24 12 12 11 10
98 98 96 87 80 80 77 75 57 57 52 51 46 36 35 33 30 29 22 16 12
87 84 73 73 71 65 64 56 55 52 47 46 45 32 22 13
98 98 94 91 89 86 83 75 67 67 65 64 59 48 48 43 40 33 25 24 24 23 16 12 12
97 89 79 72 67 56 50 46 46 44 35 35 30 29 24 15 15 10
96 95 94 91 88 74 72 63 62 60 60 52 51 40 37 31 30 28 26 22 22 22 15 13 10
98 92 85 81 77 77 73 62 52 51 49 31 27 27 27 24 20 12
87 75 71 64 56 54 48 48 47 43 41 37 35 33 31 27 25 21 20 15
87 81 79 68 67 58 56 55 49 48 46 42 40 36 33 31 30 29 25
92 88 87 85 83 81 73 64 52 48 47 46 45 44 43 32 17 13 10
80 55
averageRunLength = 19
the file is sorted: True
Max run length: 28
Min run length: 2
total Runs: 25
total items: 487

4.6 descending_randomFloatTest_out.cpp File Reference

4.6.1 Detailed Description

92 89 87 85 82 79 75 59 59 58 53 50 37 25 25 23 22 22 20 17 15 14 11 11
96 89 82 82 80 71 71 70 68 53 51 48 44 44 40 37 37 33 27 27 24 23 11
98 89 88 81 74 73 71 71 69 65 64 64 60 57 54 44 37 25 24 24 21 21 18 18
98 89 87 85 84 78 71 66 65 63 60 42 40 28 27 27 26 22 22 21 16
97 95 93 81 78 66 63 56 53 52 43 42 42 37 34 34 22 21 14 12 12
89 89 87 87 75 71 68 68 62 55 54 48 40 37 37 19 15 10
88 87 84 81 77 71 56 53 52 40 40 34 33 22 18 16 10
97 92 68 67 67 62 59 58 58 44 34 33 33 25 17 15 12
82 82 79 78 74 72 71 70 57 51 51 33 26 25 23 19 16 13
96 96 94 93 91 91 90 89 89 70 68 65 64 63 62 53 39 31 30 26 23 12
97 96 94 87 79 66 51
averageRunLength = 19

the file is sorted: True
Max run length: 24
Min run length: 7
total Runs: 11
total items: 212

4.7 descending_randomIntTest_out.cpp File Reference

4.7.1 Detailed Description

93 92 86 85 85 81 73 73 69 48 38 34 26 18 17 15 15 12 12
95 85 81 79 66 62 62 61 52 44 42 37 32 28 25
96 95 94 93 92 91 89 87 84 84 78 65 53 50 42 42 41 40 35 28 22 21 16 15 14 12
87 85 82 75 64 64 60 53 52 52 50 43 40 38 37 36 33 33 16 15 11
98 92 80 77 72 70 67 66 55 50 50 38 33 22 18 11 10
90 82 81 78 72 64 63 54 50 48 47 41 34 30 26 21 20 19 12 10
98 97 96 92 89 79 77 75 59 47 43 41 39 26 26 26 26 26 20
98 95 91 82 75 69 63 62 62 59 48 45 43 42 31 30 25 22 17 16
93 87 85 84 82 81 81 78 71 68 65 54 46 38 31 23 20
98 97 90 89 88 83 83 78 76 71 71 65 64 54 49 48 46 45 45 33 23 15 12
98 91 90 87 80 76 72 71 70 64 61 58 50 49 38 33 30 25 23 16 11
98 98 93 90 87 81 80 78 77 75 73 71 68 61 47 39 37 29 27 26 25 21 15
97 90 88 84 83 77 65 62 62 55 50 42 39 39 38 37 32 29 26 20 20 16
97 86 85 83 78 75 74 61 58 58 46 35 35 35 32 32 29 22
97 91 87 79 77 72 70 67 66 57 53 51 46 38 35 30 22 21 15 12
97 94 91 90 86 83 74 73 66 65 64 56 52 47 33 33 32 24 22
97 95 94 91 83 81 80 78 76 65 64 59 58 50 42 34 34 30 29 15 13 12 10
88 79 77 77 74 72 71 69 61 58 56 51 50 47 43 32 31 30 29 28 25 20 19 15
96 95 95 89 78 73 73 67 66 63 60 52 34 31 30 25 20 20 19 17 12
96 92 83 82 81 77 77 70 69 56 50 42 40 37 32 31 28 21 19 17 13
92 89 76 73 73 72 62 58 36 36 35 31 25 25 22 19
95 93 91 83 78 78 78 63 62 52 50 49 43 39 37 37 36 29 20
98 93 91 91 84 81 79 78 78 67 65 59 57 57 53 51 42 37 20 20 19 16 13 13 12
95 94 85 77 77 77 72 66 64 62 62 58 58 58 44 43 42 38 36 36 34 34 32 26 18 12 11 10
85 81 78 68 64 63 58 47 44 43 41 37 21 20 16 12 11 11 10
97 96 96 93 80 77 74 71 58 55 51 49 42 41 39 35 27 27 24 24 24 17
97 96 91 87 86 84 61 61 57 57 51 43 39 39 33 28 27 14 14
92 92 87 83 82 79 77 76 74 73 61 52 51 48 43 43 41 34 19 19 16 14 14 11 10
97 92 91 77 56 55 54 53 52 52 50 41 40 26 23 17 16 10
97 96 96 85 82 80 80 78 69 69 63 55 50 49 42 38 22 21 18 14
93 86 85 75 74 73 69 63 60 44 38 37 37 27 24 15
98 96 94 93 83 80 69 68 64 62 56 54 52 43 41 38 34 27 19 10
90 86 84 70 63 54 50 48 46 43 41 38 33 32 31 30 30 28 18 16
98 95 94 91 86 80 76 74 68 62 55 49 44 31 28 22 16 13 11 10
87 83 82 81 79 79 70 69 64 59 57 57 56 54 48 48 46 33 31 27 21
95 94 90 87 83 78 58 52 44 42 40 39 37 34 33 30 22 11
89 84 82 78 71 70 69 66 64 64 63 58 54 52 52 46 45 45 43 41 37 34 28 22 18 12
97 96 90 86 81 69 69 68 65 60 55 50 36 30 25 24 23 11 11 10
88 82 80 75 73 65 60 58 56 53 49 35 34 33 28 25 25 23 19 19 16 14 14 13
94 91 89 86 86 79 62 60 57 54 53 41 22 20 16 13
90 85 77 74 73 67 64 64 63 37 34 32 27 20 17 14
92 86 82 67 65
averageRunLength = 20
the file is sorted: True

Max run length: 28
Min run length: 5
total Runs: 42
total items: 842

4.8 descending_randomStringTest_out.cpp File Reference

4.8.1 Detailed Description

sE pc pK he cI ar Zq ZR Yh Ui Te Mf KH KE Jd Ex BG AA
ps oj
averageRunLength = 10
the file is sorted: True
Max run length: 18
Min run length: 2
total Runs: 2
total items: 20

4.9 DesignDocument.cpp File Reference

4.9.1 Detailed Description

Replacement Selection Sort with Tournament Merging Design Document

Introduction

This program will sort records of fixed length into small sorted records then merges them back into one record. This is done by using a dual heap to represent a fixed allocation size for the program to sort items. This process will continue until there are no more items to be sorted. This will be called the intermediate file. The intermediate file will contain each record that has been sorted. After this process is complete, the records are merged together using a tournament sort method. This will result in a single sorted record.

Data Structures

The primary data structure is the replacement selection sort algorithm internally utilizing a modified heap. As records are read from an input stream the record are inserted into the heap until the heap is full. Once the heap is full then the root is popped to an output stream. Instead of heapifying the heap the next record is taken from the input stream and if it can be put in the output stream in proper sorted order then it is placed in the root of the heap. However is the record cannot be placed in proper sort order then it is placed in a secondary heap. The two heaps share the same memory space and when a element is placed in the secondary heap the primary heap gives up space to the secondary heap. The root of the heap is popped and the next record is placed as before in the primary or secondary heap. This process continues until the the input is exhausted or the primary heap's size is zero. When the primary heap's size zero the secondary heap becomes the primary heap and a new output is started. Then the process of popping root of the primary heap and placing the next input is resumed. When the input is exhausted then the primary heap is heapify and popped to the output until it is empty. After that a new output is created to hold the records from the secondary heap. Then secondary heap's records are popped to the output and the secondary heapify until the secondary heap is empty. This result is sorted lists of records.

Before the Tournament Sort the Program will produce not just a sorted list but also a vectors of vectors that it will then copy the information of the sorted list onto the vector of vectors. From there the tournament sort will begin as

it take's the data from the vector of vectors and pushes it into a priority queue. Where from there the it will continue to push and pop the priority queue the information from largest to smallest (depending on desired sort order) into the final vector which will then print out as a single sorted list.

Functions

Functions in main.cpp.

test(string outputFileName, bool direction_flag) is a function that is only there to test the selection Sort Process.

generateTestFile(ostream& outfile) is a function that randomly generates numbers to a file. vector<vector<↩ T> checkRuns(istream& infile, ostream& outfile, bool direction_flag) is a vector function that Evaluates various information about the selection sort after the selection sort process is complete.

stringTest(string inputFileName, string outputFileName, bool ascending) is a specfic function that applies replacement selection sort on strings.

randomStrings(ostream& outfile) is a function that generates random strings for the output file.

gen_random(char *s, const int len) is a function that generates random character.

Functions in ReplacementSelectionSort.cpp.

ReplacementSelectSort(const int _size, istream& _infile, ostream& _outputfile, const bool _sort) which is a constructor of the class.

~ReplacementSelectionSort Destructor

sort(istream& infile, ostream& outputfile) which use's replacement selection sort to create multiple sorted lists.

heapify(bool leftSide) which builds the heap structure.

siftUp(int index, bool leftSide) is a function that has operations to insert an element.

siftDown(int index, bool leftSide) is a function that use's operation that remove an element.

int left(int index, bool leftSide) is a function that use's operations to find and return the left indice of a given element.

int right(int index, bool leftSide) is a function that use's operations that find and return the right indice of a given element.

int parent(int index, bool leftSide) uses operations that find and return the parent indicie of a given element.

void swap(int index1, int index2) is a function that swaps two elements.

T pop() is a function that pops the top element in the heap.

T initHeap(istream& infile) is a function that initializes the heap.

current_dualHeap_push(const T entry) is a function that pushes an element onto the active/current side of the dual heap.

pending_dualHeap_push(const T entry) is a function that pushes an element onto the pending side of the dual heap.

void swapActive() is a function that swaps which element is an active and which are pending sides.

void sortPreMerge(istream* infile, ostream& outputfile) is a helper function for sorting that uses the replacement selection sort algorithm.

Functions in TournamentSort.h.

Node(int x, int y, T z) {position = x; rn = y; value = z;} is a constructor for the class Node that holds the position the value is in. It's ins respective list it also holds the relative record number and the raw data that is being stored.

Virtual ∼Node() is a default destructor.

TournamentSort() is a Default constructor.

Virtual ∼TournamentSort() is a Default destructor.

std::priority_queue<Node, std::vector<Node>, compare> initPQ(std::priority_queue<Node, std::vector<Node>, compare> PQ, std::vector<vector<T> > sortedLists) is a function that initializes the priority queue.

Void pushToFinal(std::priority_queue <Node, std::vector<Node>, compare> PQ, vector<T> final, std↩ ::vector<vector<T> > sortedLists) is a function that pushes the smallest item in the priority queue and then it re-populates it.

The Main Program

A seed is hardcoded to use for the pseudorandom generator. Each test data for different variable types will contain a variable number of records. At runtime, the user will enter a flag to let the program know what type of data they are sorting. This flag can be -string, -int, -float, -double, or -all.The program will check the flag through a switch statement to determine the correct sorting that needs to be conducted. Tests can be run by the command line. Flags are as such: -i -a will sort ints ascending, -i with no second flag with default to descending sorting. This convention is continued for strings, doubles and floats; using the first letter of the data type as the first flag and the second flag as the sorting order. The last single flag is -a which runs all of the tests. Each input data that will be tested with then procede to be sorted by replacement selection sort then merged using tournament sort. The results will be stored in a final output file. A relative record number file will be generated for the last test run. This file will contain three fields; the RRN, size of the run and the run number. The results file will contain the sorted list. If all tests are run, this fill will concatenate the results from all of the tests.

4.10 mainProgram.cpp File Reference

```
#include "ReplacementSelectionSort.h"
#include "ReplacementSelectionSort.cpp"
#include "TournamentSort.h"
#include <fstream>
#include <iostream>
#include <string>
#include <typeinfo>
#include <random>
#include <cmath>
#include <sstream>
#include <stdio.h>
#include <queue>
#include <time.h>
#include <algorithm>
#include <iterator>
#include <climits>
#include <functional>
#include <string.h>
```

Include dependency graph for mainProgram.cpp:

Functions

- template<typename T >
  void test (string input, string output, bool direction_flag, string typeUsed, bool reverse)
- template<typename T >
  void generateTestFile (ostream &outfile)
- template<typename T >
  vector< vector< T > > checkRuns (istream &infile, ostream &outfile, bool direction_flag)
- void stringTest (string inputFileName, string outputFileName, bool ascending)
- void randomStrings (ostream &outfile)
- void gen_random (char *s, const int len)
- void generateTestFileFloat (ostream &outfile)
- int main (int argc, char *argv[])
- void merge (int a[], int startIndex, int endIndex)

4.10.1 Function Documentation

4.10.1.1 checkRuns()

```
template<typename T >
vector< vector< T > > checkRuns (
            istream & infile,
            ostream & outfile,
            bool direction_flag )
```

Evaluate various information about the selection sort after the selection sort process is complete

Parameters

infile	input file
outfile	output file
direction_flag	True = ascending

4.10.1.2 gen_random()

```
void gen_random (
            char * s,
            const int len )
```

Generates random characters

Parameters

s	output character
len	length of c-string

4.10.1.3 generateTestFile()

```
template<typename T >
```

```
void generateTestFile (
            ostream & outfile )
```

Function to randomly generage numbers to a file

Parameters

outfile	File that will hold the randomly generated numbers

4.10.1.4 generateTestFileFloat()

```
void generateTestFileFloat (
            ostream & outfile )
```

Generate random floats

Parameters

outfile	file to write random floats to

4.10.1.5 main()

```
int main (
            int argc,
            char * argv[] )
```

4.10.1.6 merge()

```
void merge (
            int a[],
            int startIndex,
            int endIndex )
```

4.10.1.7 randomStrings()

```
void randomStrings (
            ostream & outfile )
```

Generates random strings

Parameters

outfile	output file

4.10.1.8 stringTest()

```
void stringTest (
             string inputFileName,
             string outputFileName,
             bool ascending )
```

Specific function to apply replacement selection sort on strings

Parameters

inputFileName	input file to be tested
outputFileName	output file
ascending	True = ascending

4.10.1.9 test()

```
template<typename T >
void test (
             string input,
             string output,
             bool direction_flag,
             string typeUsed,
             bool reverse )
```

Function to test only the selection sort process

Parameters

inputFileName	The placeholder for the file for input. This can contain any data because it will be overwritten
outputFileName	The output file
direction_flag	True = sort ascending
reverse	reverses the output when merging in ascenging order

4.11 makefile.cpp File Reference

4.11.1 Detailed Description

CXX = g++
CXXFLAGS = -std=c++0X

mainProgram: mainProgram.cpp ReplacementSelectionSort.cpp ReplacementSelectionSort.h TournamentSort.h
-o mainProgram mainProgram.o ReplacementSelectionSort.o TournamentSort.o

4.12 out.dox File Reference

4.13 output.cpp File Reference

4.13.1 Detailed Description

AB AZ AZ Ar Bh CE CY DW Da EA Eb FD Gb Gb Gh Gs HD Hu IM JI JL JW MM MT NM Pa Pf Pi QR VZ Vt Vy Wi Yq aU bW bv cB cY dK dT dy eG eS gn iq jE ji kp lS mj nF ot ra rb rb sK tX tq vR wv xx zi zv

End of test string Ascending = true sE ps pc pK oj oj he cI ar Zq ZR Yh Ui Te Mf KH KE Jd Ex BG AA

End of test string Ascending = false 10 10 10 10 10 11 11 11 11 13 13 13 13 13 14 15 15 16 17 19 19 22 22 24 25 25 25 25 25 25 26 26 26 26 27 27 27 27 27 28 28 29 29 29 32 32 32 33 33 33 33 35 35 35 36 36 36 37 37 37 39 42 42 43 44 44 44 45 45 46 46 46 47 47 47 48 48 48 49 49 50 51 52 53 53 53 54 54 55 55 56 56 57 58 58 59 59 60 60 61 61 61 61 62 62 62 63 64 65 66 67 67 67 67 68 69 69 70 70 71 71 72 72 72 72 74 74 74 75 75 75 75 75 76 76 76 77 78 79 79 79 79 79 80 80 80 80 81 81 82 82 83 84 84 85 85 85 85 87 87 87 87 87 87 88 88 89 89 89 90 91 91 92 93 93 93 94 95 95 97 98 98 98

End of test integersAscending = 1 98 98 98 98 98 98 98 98 98 98 97 97 97 97 97 97 97 97 97 97 97 97 96 96 96 96 96 96 96 96 96 96 96 95 95 95 95 95 95 95 95 95 95 94 94 94 94 94 94 94 94 93 93 93 93 93 93 93 93 93 92 92 92 92 92 92 92 92 92 92 91 91 91 91 91 91 91 91 91 91 91 90 90 90 90 90 90 90 90 90 90 89 89 89 89 89 89 88 88 88 88 87 87 87 87 87 87 87 87 87 87 86 86 86 86 86 86 86 86 86 86 86 85 85 85 85 85 85 85 85 85 85 84 84 84 84 84 84 84 84 83 83 83 83 83 83 83 83 83 83 83 82 82 82 82 82 82 82 82 82 82 81 81 81 81 81 81 81 81 81 81 80 80 80 80 80 80 80 80 80 79 79 79 79 79 79 79 79 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 76 76 76 76 76 76 75 75 75 75 75 75 74 74 74 74 74 74 74 73 73 73 73 73 73 73 73 73 73 73 73 72 72 72 72 72 72 72 71 71 71 71 71 71 71 71 70 70 70 70 70 70 70 69 69 69 69 69 69 69 69 69 69 69 69 68 68 68 68 68 68 67 67 67 67 67 67 66 66 66 66 66 66 66 65 65 65 65 65 65 65 65 65 65 65 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 63 63 63 63 63 63 63 63 63 63 62 62 62 62 62 62 62 62 62 62 62 62 62 61 61 61 61 61 61 61 61 60 60 60 60 60 60 59 59 59 59 59 58 58 58 58 58 58 58 58 58 58 58 58 58 58 57 57 57 57 57 57 57 57 56 56 56 56 56 56 56 55 55 55 55 55 55 55 54 54 54 54 54 54 54 54 54 53 53 53 53 53 53 53 52 52 52 52 52 52 52 52 52 52 52 52 52 51 51 51 51 51 51 50 50 50 50 50 50 50 50 50 50 50 50 50 50 49 49 49 49 49 49 49 48 48 48 48 48 48 48 48 47 47 47 47 47 47 46 46 46 46 46 46 46 45 45 45 45 45 44 44 44 44 44 44 44 43 43 43 43 43 43 43 43 43 43 43 43 43 42 42 42 42 42 42 42 42 42 42 42 42 42 41 41 41 41 41 41 41 41 41 41 41 40 40 40 40 40 39 39 39 39 39 39 39 39 39 38 38 38 38 38 38 38 38 38 38 38 38 38 38 37 37 37 37 37 37 37 37 37 37 37 37 37 37 36 36 36 36 36 36 36 35 35 35 35 35 35 35 35 34 34 34 34 34 34 34 34 34 34 34 34 34 33 33 33 33 33 33 33 33 33 33 33 33 33 32 32 32 32 32 32 32 32 32 32 31 31 31 31 31 31 31 31 30 30 30 30 30 30 30 30 30 30 30 29 29 29 29 29 29 28 28 28 28 28 28 28 28 28 27 27 27 27 27 27 27 26 26 26 26 26 26 26 26 26 26 26 25 25 25 25 25 25 25 25 25 25 25 25 24 24 24 24 24 24 23 23 23 23 23 23 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 21 21 21 21 21 21 21 21 21 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 19 19 19 19 19 19 19 19 19 19 19 19 18 18 18 18 18 18 17 17 17 17 17 17 17 17 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 15 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13 13 13 13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 11 11 11 11 11 11 11 11 11 11 11 11 11 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

End of test integersAscending = 0 10 10 10 10 11 11 12 12 12 12 12 12 12 12 13 13 13 14 14 14 14 14 14 14 14 15 15 15 15 15 15 16 16 16 16 17 17 18 18 18 18 19 19 19 20 20 20 21 21 21 21 21 22 23 23 23 23 24 24 24 24 25 25 25 25 25 26 26 26 27 27 28 28 28 28 28 29 29 29 30 31 31 31 32 32 32 32 33 33 33 33 33 34 34 34 35 35 35 36 36 36 37 38 38 39 40 40 40 41 41 42 42 43 43 43 43 43 43 43 43 44 44 44 45 45 46 46 47 47 47 47 48 48 48 48 48 49 49 49 49 50 50 51 51 51 51 51 51 52 52 52 53 53 53 53 53 53 54 54 54 55 55 55 55 56 56 57 57 57 57 58 58 59 59 59 59 59 59 60 61 61 62 62 62 62 63 63 63 63 64 64 64 65 65 66 66 66 66 66 67 67 67 68 68 68 69 69 69 70 70 71 72 72 72 72 73 73 74 74 74 75 75 75 75 76 76 76 76 77 77 77 77 77 77 77 77 78 78 78 79 79 80 80 80 80 82 82 82 82 83 83 83 84 85 85 85 85 86 86 86 86 87 87 87 87 87 87 88 88 88 88 89 89 90 90 91 91 91 91 92 92 92 92 93 93 93 93 94 94 94 94 94 94 95 95 95 95 95 95 97 97 98 98

End of test floatsAscending = 1 98 98 97 97 97 96 96 96 96 95 94 94 93 93 92 92 91 91 90 89 89 89 89 89 89 89 89 88 88 87 87 87 87 87 87 85 85 84 84 82 82 82 82 82 81 81 81 80 79 79 79 78 78 78 77 75 75 74 74 73 72 71

71 71 71 71 71 71 71 70 70 70 69 68 68 68 68 68 67 67 66 66 66 65 65 65 64 64 64 63 63 63 62 62 62 60 60 59 59 59 58 58 58 57 57 56 56 55 54 54 53 53 53 53 53 52 52 51 51 51 51 51 50 48 48 44 44 44 44 43 42 42 42 40 40 40 40 40 39 37 37 37 37 37 37 37 34 34 34 34 33 33 33 33 33 31 30 28 27 27 27 27 26 26 26 25 25 25 25 25 24 24 24 23 23 23 23 22 22 22 22 22 22 21 21 21 21 20 19 19 18 18 18 18 17 17 16 16 16 16 15 15 15 14 14 13 13 12 12 12 12 12 12 12 11 11 11 11 11 10 10 10

End of test floatsAscending = 0 10 10 10 10 10 10 11 11 12 12 12 12 12 12 13 13 13 13 13 13 14 14 14 14 14 14 16 16 16 17 17 17 17 17 17 18 18 18 18 18 18 19 19 20 21 21 21 22 22 23 23 23 24 24 24 26 27 27 28 28 28 28 28 28 29 30 30 30 31 31 31 31 31 31 31 31 32 32 32 32 32 32 33 34 34 35 36 36 37 38 38 38 38 40 40 41 41 41 41 41 42 42 43 44 44 44 45 45 45 46 46 46 46 46 46 47 47 47 48 48 48 48 48 48 49 49 50 50 50 50 51 52 52 52 53 54 55 55 55 55 55 55 56 56 56 57 57 57 59 59 60 60 60 61 62 62 63 63 64 65 65 65 66 66 66 66 66 67 67 67 68 69 71 71 71 73 73 73 73 73 74 74 74 75 76 76 77 77 77 78 78 79 79 80 80 81 82 83 83 83 84 84 84 85 85 86 86 87 87 88 88 89 90 91 91 92 92 93 93 95 95 95 96 96 97 98 98 98

End of test doublesAscending = 1 98 98 98 98 98 98 98 98 98 97 97 97 97 97 97 97 97 96 96 96 96 96 95 95 95 94 94 94 94 93 93 93 93 93 93 93 92 92 92 92 92 92 92 92 92 91 91 91 91 91 90 90 90 90 90 90 89 89 89 89 89 89 88 88 87 87 87 87 87 87 87 87 86 86 86 86 86 85 85 85 85 85 84 84 84 83 83 83 83 83 82 82 82 82 82 82 81 81 81 81 81 81 81 81 80 80 80 80 80 80 80 79 79 79 79 79 78 78 78 77 77 77 77 77 77 77 77 77 77 77 76 76 76 76 76 76 75 75 75 75 75 74 74 73 73 73 73 73 73 72 72 72 72 71 71 71 70 70 69 69 69 69 68 68 68 68 68 67 67 67 67 67 67 67 66 66 66 66 66 66 66 65 65 65 65 65 64 64 64 64 64 63 63 63 63 63 63 62 62 62 62 62 62 62 62 61 61 61 60 60 60 59 59 59 59 59 59 58 58 58 57 57 57 57 57 57 57 57 56 56 56 56 56 56 56 56 55 55 55 55 55 55 54 54 54 53 53 53 53 53 52 52 52 52 52 52 52 52 52 51 51 51 51 51 51 51 51 50 50 50 49 49 49 49 49 48 48 48 48 48 48 48 48 48 48 47 47 47 47 47 47 47 47 46 46 46 46 46 46 46 46 45 45 45 45 45 44 44 44 43 43 43 43 43 42 42 41 41 41 41 41 40 40 40 40 39 39 39 39 39 38 38 38 37 37 37 37 36 36 36 36 36 35 35 35 35 35 35 35 35 35 34 34 33 33 33 33 33 33 33 33 33 33 33 33 32 32 32 32 32 31 31 31 31 31 31 31 31 31 31 30 30 30 30 30 30 29 29 29 29 29 28 28 28 28 28 28 28 27 27 27 27 27 27 27 26 26 26 25 25 25 25 25 25 25 25 25 24 24 24 24 24 24 24 23 22 22 22 22 22 21 21 21 20 20 20 20 20 20 19 19 19 19 18 18 18 18 18 18 18 18 18 17 17 17 17 17 16 16 15 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 13 13 13 13 13 13 13 13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 11 10 10 10 10 10 10 10 10 10 10 10 10

End of test doublesAscending = 0

4.14 randomDoubleTest_in.txt File Reference

4.15 randomfloatTest_in.txt File Reference

4.16 randomIntTest_in.txt File Reference

4.17 randomStringTest_in.txt File Reference

4.18 ReplacementSelectionSort.cpp File Reference

A class to apply Replacement Selection Sort to a file of unsorted items.

```
#include "ReplacementSelectionSort.h"
#include <string>
```

Include dependency graph for ReplacementSelectionSort.cpp: This graph shows which files directly or indirectly include this file:

4.18.1 Detailed Description

A class to apply Replacement Selection Sort to a file of unsorted items.

4.19 ReplacementSelectionSort.h File Reference

A class to apply Replacement Selection Sort to a file of unsorted items.

```
#include <vector>
#include <iostream>
#include <fstream>
#include <functional>
#include <typeinfo>
```

Include dependency graph for ReplacementSelectionSort.h: This graph shows which files directly or indirectly include this file:

Classes

- class ReplacementSelectionSort< T >

4.19.1 Detailed Description

A class to apply Replacement Selection Sort to a file of unsorted items.

4.20 rrnFile.cpp File Reference

4.20.1 Detailed Description

rn: 15 size: 15 runNum: 1
rn: 36 size: 21 runNum: 2
rn: 60 size: 24 runNum: 3
rn: 78 size: 18 runNum: 4
rn: 101 size: 23 runNum: 5
rn: 122 size: 21 runNum: 6
rn: 142 size: 20 runNum: 7
rn: 164 size: 22 runNum: 8
rn: 180 size: 16 runNum: 9
rn: 198 size: 18 runNum: 10
rn: 226 size: 28 runNum: 11
rn: 244 size: 18 runNum: 12
rn: 265 size: 21 runNum: 13
rn: 285 size: 20 runNum: 14
rn: 304 size: 19 runNum: 15
rn: 325 size: 21 runNum: 16
rn: 341 size: 16 runNum: 17
rn: 366 size: 25 runNum: 18
rn: 384 size: 18 runNum: 19
rn: 409 size: 25 runNum: 20
rn: 427 size: 18 runNum: 21
rn: 447 size: 20 runNum: 22
rn: 466 size: 19 runNum: 23
rn: 485 size: 19 runNum: 24
rn: 487 size: 2 runNum: 25

4.21 SourceCode.cpp File Reference

4.21.1 Detailed Description

#include "ReplacementSelectionSort.h" #include "ReplacementSelectionSort.cpp" #include "TournamentSort.h" #include <fstream> #include <iostream> #include <string> #include <typeinfo> #include <random> #include <cmath> #include <sstream> #include <stdio.h> #include <queue> #include <time.h> #include <algorithm> #include <iterator> #include <climits> #include <queue> #include <functional> #include <string> #include <string.h>

using namespace std;

template <typename t>=""> void test(string input, string output, bool direction_flag, string typeUsed, bool reverse);

template<typename T> void generateTestFile(ostream& outfile);

template <typename t>=""> vector< vector<T> > checkRuns(istream& infile, ostream& outfile, bool direction↩ _flag);

void stringTest(string inputFileName, string outputFileName, bool ascending);

void randomStrings(ostream& outfile);

void gen_random(char *s, const int len);

void generateTestFileFloat(ostream& outfile);

int main(int argc, char* argv[ ]) { srand(11);

```
if(argc > 1){

    switch(argv[1][1]){

        case 's':if(argc > 2) { //string && ascending


                        string inputFileName = "randomStringTest_in.txt";
                        string outputFileName = "ascending_randomStringTest_out";
                        cout << "starting\n"<<std::flush;
                        ofstream testFile(inputFileName.c_str());
                        cout << "Creating test file of Strings\n"<<std::flush;
                        randomStrings(testFile);
                        testFile.close();

                        ifstream inputFile(inputFileName.c_str());
                        bool ascending = false;
                        ofstream otemp("temp.txt");
                        cout<<"replacement selection\n"<<std::flush;
                        ReplacementSelectionSort<string> sortedRuns(10, inputFile, otemp, ascending);
                        ofstream outputFile(outputFileName.c_str());
                        sortedRuns.sort(inputFile, otemp);
                        otemp.close();

                        cout << "checking runs\n"<<std::flush;
                        ifstream itemp("temp.txt");
```

/ofstream outputFile(outputFileName); vector< vector<string> > listOfLists;

listOfLists = checkRuns<string> (itemp, outputFile, ascending); cout << "got the list\n"<<std::flush; vector<vector<string>> listOfLists = {second, third, first, fourth, seventh, sixth, fifth}; vector<string> final; TournamentSort<string> tournament;

std::priority_queue<TournamentSort<string>::Node, std::vector<TournamentSort<string>::Node>,Tournament↩ Sort<string>::compare> PQ;

cout<<"about to initialize\n"<<std::flush; PQ = tournament.initPQ(PQ, listOfLists); cout<<"about to sortmerge\n"<<std::flush; tournament.pushToFinal(PQ, final, listOfLists); cout<<"about to write\n"<<std::flush; ofstream myfile; myfile.open ("output.txt", ios::app); for(int i = tournament.finalVector.size()-1; i>=0; i–) { myfile << tournament.finalVector[i]<<"\n";

}

myfile << "\n---\n"; myfile << "\nEnd of test string Ascending = false\n" ; myfile.close();

```
                }else{// string && descending



                    string inputFileName = "randomStringTest_in.txt";
                    string outputFileName = "descending_randomStringTest_out";
                    cout << "starting\n"<<std::flush;
                    ofstream testFile(inputFileName.c_str());
                    cout << "Creating test file of Strings\n"<<std::flush;
                    randomStrings(testFile);
                    testFile.close();

                    ifstream inputFile(inputFileName.c_str());
                    bool ascending = false;
                    ofstream otemp("temp.txt");
                    cout<<"replacement selection\n"<<std::flush;
                    ReplacementSelectionSort<string> sortedRuns(10, inputFile, otemp, ascending);
                    ofstream outputFile(outputFileName.c_str());
                    sortedRuns.sort(inputFile, otemp);
                    otemp.close();

                    cout << "checking runs\n"<<std::flush;
                    ifstream itemp("temp.txt");
```

/ofstream outputFile(outputFileName); vector< vector<string> > listOfLists;

listOfLists = checkRuns<string> (itemp, outputFile, ascending); cout << "got the list\n"<<std::flush; vector<vector<string>> listOfLists = {second, third, first, fourth, seventh, sixth, fifth}; vector<string> final; TournamentSort<string> tournament;

std::priority_queue<TournamentSort<string>::Node, std::vector<TournamentSort<string>::Node>,Tournament↩ Sort<string>::compare> PQ;

cout<<"about to initialize\n"<<std::flush; PQ = tournament.initPQ(PQ, listOfLists); cout<<"about to sortmerge\n"<<std::flush; tournament.pushToFinal(PQ, final, listOfLists); cout<<"about to write\n"<<std↩ ::flush; ofstream myfile; myfile.open ("output.txt", ios::app); for(int i = 0; i < tournament.finalVector.size()-1; i++) { myfile << tournament.finalVector[i]<<"\n";

}

myfile << "\n---\n"; myfile << "\nEnd of test string Ascending = true\n" ; myfile.↩ close();

```
                }//END ELSE
                break;

    case 'i':if(argc > 2){ //INT && ASCENDING

                test<int>("randomIntTest_in.txt", "ascending_randomIntTest_out", false, "integers", true); //
            }else{
                test<int>("randomIntTest_in.txt", "descending_randomIntTest_out", false, "integers", false);
            }
            break;
    case 'f':if(argc > 2){ //FLOAT && ASCENDING

                 test<float>("randomFloatTest_in.txt", "ascending_randomFloatTest_out", false, "floats", true)
            }else{
                 test<float>("randomFloatTest_in.txt", "descending_randomFloatTest_out", false, "floats", fals
            }
            break;
    case 'd':if(argc > 2){ //DOUBLE && ASCENDING

                test<double>("randomDoubleTest_in.txt", "ascending_randomDoubleTest_out", false, "doubles", tr
            }else{
                test<double>("randomDoubleTest_in.txt", "descending_randomDoubleTest_out", false, "doubles", f
            }
            break;
    case 'a': // RUN ALL TESTS



                        for(int i = 0 ; i < 1; i++){ //Keep variables in an isolated scope

                        string inputFileName = "randomStringTest_in.txt";
                        string outputFileName = "ascending_randomStringTest_out";
                        cout << "starting\n"<<std::flush;
                        ofstream testFile(inputFileName.c_str());
                        cout << "Creating test file of Strings\n"<<std::flush;
                        randomStrings(testFile);
                        testFile.close();

                        ifstream inputFile(inputFileName.c_str());
                        bool ascending = false;
                        ofstream otemp("temp.txt");
                        cout<<"replacement selection\n"<<std::flush;
                        ReplacementSelectionSort<string> sortedRuns(10, inputFile, otemp, ascending);
                        ofstream outputFile(outputFileName.c_str());
                        sortedRuns.sort(inputFile, otemp);
                        otemp.close();

                        cout << "checking runs\n"<<std::flush;
                        ifstream itemp("temp.txt");
```

/ofstream outputFile(outputFileName); vector <vector<string> > listOfLists;

listOfLists = checkRuns<string> (itemp, outputFile, ascending); cout << "got the list\n"<<std::flush; vector<vector<string>> listOfLists = {second, third, first, fourth, seventh, sixth, fifth}; vector<string> final; TournamentSort<string> tournament;

std::priority_queue<TournamentSort<string>::Node, std::vector<TournamentSort<string>::Node>,Tournament↩ Sort<string>::compare> PQ;

cout<<"about to initialize\n"<<std::flush; PQ = tournament.initPQ(PQ, listOfLists); cout<<"about to sortmerge\n"<<std::flush; tournament.pushToFinal(PQ, final, listOfLists); cout<<"about to write\n"<<std::flush; ofstream myfile; myfile.open ("output.txt", ios::app); for(int i = tournament.finalVector.size()-1; i>=0; i–) { myfile << tournament.finalVector[i]<<"\n";

} myfile << "\n----------\n"; myfile << "\nEnd of test string Ascending = true\n" ; myfile.close();

}

//////////////////////////////////////

```
                        for(int i = 0 ; i < 1; i++){ //Keep variables in an isolated scope

                        string inputFileName = "randomStringTest_in.txt";
                        string outputFileName = "descending_randomStringTest_out";
                        cout << "starting\n"<<std::flush;
                        ofstream testFile(inputFileName.c_str());
                        cout << "Creating test file of Strings\n"<<std::flush;
                        randomStrings(testFile);
                        testFile.close();

                        ifstream inputFile(inputFileName.c_str());
                        bool ascending = false;
                        ofstream otemp("temp.txt");
                        cout<<"replacement selection\n"<<std::flush;
                        ReplacementSelectionSort<string> sortedRuns(10, inputFile, otemp, ascending);
                        ofstream outputFile(outputFileName.c_str());
                        sortedRuns.sort(inputFile, otemp);
                        otemp.close();

                        cout << "checking runs\n"<<std::flush;
                        ifstream itemp("temp.txt");
```

/ofstream outputFile(outputFileName); vector <vector<string> > listOfLists;

listOfLists = checkRuns<string> (itemp, outputFile, ascending); cout << "got the list\n"<<std::flush; vector<vector<string>> listOfLists = {second, third, first, fourth, seventh, sixth, fifth}; vector<string> final; TournamentSort<string> tournament;

std::priority_queue<TournamentSort<string>::Node, std::vector<TournamentSort<string>::Node>,Tournament↩ Sort<string>::compare> PQ;

cout<<"about to initialize\n"<<std::flush; PQ = tournament.initPQ(PQ, listOfLists); cout<<"about to sortmerge\n"<<std::flush; tournament.pushToFinal(PQ, final, listOfLists); cout<<"about to write\n"<<std↩ ::flush; ofstream myfile; myfile.open ("output.txt", ios::app); for(int i = 0; i < tournament.finalVector.size()-1; i++) { myfile << tournament.finalVector[i]<<"\n";

} myfile << "\n----------\n"; myfile << "\nEnd of test string Ascending = false\n" ; myfile.close();

}

//////////////////////////////////////

```
                    test<int>("randomIntTest_in.txt", "ascending_randomIntTest_out", false, "integers", true);
                    test<int>("randomIntTest_in.txt", "descending_randomIntTest_out", false, "integers", false
                    test<float>("randomFloatTest_in.txt", "ascending_randomFloatTest_out", false, "floats", tr
                    test<float>("randomFloatTest_in.txt", "descending_randomFloatTest_out", false, "floats", f
                    test<double>("randomDoubleTest_in.txt", "ascending_randomDoubleTest_out", false, "doubles"
                    test<double>("randomDoubleTest_in.txt", "descending_randomDoubleTest_out", false, "doubles

                    break;



    }//END SWITCH

} // END IF ARGC > 1



return (0);
```

}

template <typename t>=""> void test(string input, string output, bool direction_flag, string typeUsed, bool reverse) { string inputFileName = input; string outputFileName = output; cout << "starting\n"<<std::flush; ofstream testFile(inputFileName.c_str());

cout << "Creating test file "<<typeUsed <<"\n"<<std::flush; if(typeUsed.compare("float") == 0){

```
generateTestFileFloat(testFile);
```

}else{

```
generateTestFile<T>(testFile);
```

} testFile.close();

ifstream inputFile(input.c_str()); bool ascending = direction_flag; ofstream otemp("temp.txt"); cout<<"replacement selection\n"<<std::flush; ReplacementSelectionSort<string> sortedRuns(10, inputFile, otemp, ascending); ofstream outputFile(output.c_str()); sortedRuns.sort(inputFile, otemp); otemp.close();

cout << "checking runs\n"<<std::flush; ifstream itemp("temp.txt"); /ofstream outputFile(outputFileName); vector <vector<T> > listOfLists;

listOfLists = checkRuns<T> (itemp, outputFile, ascending); cout << "got the list\n"<<std::flush; vector<vector<string>> listOfLists = {second, third, first, fourth, seventh, sixth, fifth}; vector<T> final; TournamentSort<T> tournament;

std::priority_queue<typename TournamentSort<T>::Node, std::vector<typename TournamentSort<T>::Node>, typename TournamentSort<T>::compare> PQ;

```
cout<<"about to initialize\n"<<std::flush;
PQ = tournament.initPQ(PQ, listOfLists);
cout<<"about to sortmerge\n"<<std::flush;
tournament.pushToFinal(PQ, final, listOfLists);
cout<<"about to write\n"<<std::flush;
ofstream myfile;
myfile.open ("output.txt", ios::app);
if (!reverse)
{
    for(int i = 0; i < tournament.finalVector.size()-1; i++)
    {
        myfile << tournament.finalVector[i]<<"\n";

    }
}
else
{
    for(int i = tournament.finalVector.size()-1; i>=0; i--)
                        {
                            myfile << tournament.finalVector[i]<<"\n";

                        }
}


myfile << "\n-----------------------------------------------\n";
myfile << "\nEnd of test " << typeUsed << "Ascending = " << reverse<<"\n";
myfile.close();
```

}

template<typename T> void generateTestFile(ostream& outfile) {

```
int strlength;
int fileLen;
fileLen = rand() % (rand() % 1000);

for(int i = 0; i < fileLen; i++)
{

    strlength = 2;

    T num = (rand() % 89) + 10;


    outfile << num << " ";

    if(i % 5 == 0)
    {

        outfile << "\n";
        }

    }


}
```

void generateTestFileFloat(ostream& outfile){

```
int strlength;
int fileLen;
fileLen = rand() % (rand() % 1000);

for(int i = 0; i < fileLen; i++)
{

    strlength = 2;

    float num = (rand() / (float)RAND_MAX * 99) + 1;


    outfile << num << " ";

    if(i % 5 == 0)
    {

        outfile << "\n";
        }

    }


}
```

template <typename t>=""> vector <vector<T> > checkRuns(istream& infile, ostream& outfile, bool direction↩
_flag) {

```
cout << "in check run\n"<<std::flush;
int runLength = 0;
int minLength = 0;

int maxRunLength = 0;
int minRunLength = 0;
int outRunNumber;

int totalObj = 0;

int totalRuns = 0;

bool isSorted = true;

int rn = 0;
int size = 0;
```

```
int runNum = 0;

int prev_size = 0;

int prevRN = 0;

T prevObj;
T curObj;

string curLine;

vector <vector<T> > listOfLists;

ofstream outClean("out.txt");
ofstream myfile;
    myfile.open("rrnFile.text");
cout << "opened the file\n"<<std::flush;
while(getline(infile, curLine))
{
    cout << "got a line\n"<<std::flush;
    vector<T> currentRun;
    runLength = 0;
    outfile << curLine << "\n";
    outClean << curLine << "\n";
    totalRuns++;
    istringstream stringStream(curLine);
    stringStream >> prevObj;
    currentRun.push_back(prevObj);
    runLength++;
```

totalObj++;

```
    runNum++;

    while(!stringStream.eof())
    {

        stringStream >> curObj;
        currentRun.push_back(curObj);
        runLength++;
        totalObj++;
        rn = totalObj;
        size = runLength - 1;


        if(direction_flag)
        {

            if(curObj < prevObj)
            {

                isSorted = false;
            }
        }
        else
        {

            if(curObj > prevObj)
            {
                isSorted = false;
            }

        }


    }
    cout << "added one list\n"<<std::flush;
    listOfLists.push_back(currentRun);
    if(runLength > maxRunLength)
    {
        maxRunLength = runLength;
    }
```

```
    if(size < maxRunLength){
        minRunLength = size;
    }

    if(prevRN != rn){
            myfile << "rn:  " << rn << "  size:  " << size << "  runNum:  " << runNum << "\n";
            outRunNumber = runNum;
    }

    prevRN = rn;


}



outfile
        <<"averageRunLength = " << totalObj / (  outRunNumber)
        <<"\nthe file is sorted: "<< (isSorted? "True" : "False")
        <<"\nMax run length: " << maxRunLength - 1
        <<"\nMin run length: " << minRunLength
        <<"\ntotal Runs: " << outRunNumber
        <<"\ntotal items: " << totalObj;

cout << "returning the list\n"<<std::flush;
return listOfLists;


}
```

void randomStrings(ostream& outfile) {

```
int strlength = 0;
int fileLen;

fileLen = rand() % (100 + rand() % 100);


for(unsigned int i = 0; i < fileLen; i++)
{

    strlength = 2;
    char* str = (char*) malloc(strlength * (sizeof(char)));
    gen_random(str,strlength);

    outfile << str << " ";

    if( i % 5 == 0 )
    {
        outfile << "\n";
    }

}


}
```

void gen_random(char *s, const int len) {

```
static const char alphanum[] =

    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    "abcdefghijklmnopqrstuvwxyz";

for (int i = 0; i < len; ++i)
{
    s[i] = alphanum[rand() % (sizeof(alphanum) - 1)];
}

s[len] = 0;
```

}

#ifndef REPLACEMENT_SELECTION_SORT #define REPLACEMENT_SELECTION_SORT #include <vector> #include <iostream> #include <fstream> #include <functional> #include <typeinfo>

using namespace std;

template <class t=""> class ReplacementSelectionSort { public:

```
ReplacementSelectionSort(const int _size, istream& _infile, ostream& _outputfile,  const bool _sort);

virtual ~ReplacementSelectionSort();
```

protected: private: bool activeLeftHeap; bool direction_flag; int size; int leftHeapStart; int leftHeapEnd; int right↩ HeapStart; int rightHeapEnd;

```
vector<T> dualHeap;


void sort(istream& infile, ostream& outputfile);

void heapify (bool leftSide);

void siftUp(int index, bool leftSide);

void siftDown (int index, bool leftSide);


int left(int index, bool leftSide);


int right(int index, bool leftSide);

int parent(int index, bool leftSide);
void swap(int index1, int index2);


T pop();

T initHeap(istream& infile);


void current_dualHeap_push(const T entry);
void pending_dualHeap_push (const T entry);

void swapActive();


void sortPreMerge(istream& infile, ostream& outputfile);
```

};

#endif // REPLACEMENTSELECTIONSORT_H

#include "ReplacementSelectionSort.h" #include <string>

Constructor for the ReplacementSelectionSort class. The sets the input file, output file, the is of the memory, and the sort direction. As well as initialize the heaps and sorting the data. template <typename t>=""> ReplacementSelectionSort<T>::ReplacementSelectionSort(const int _size, istream& _infile, ostream& _outputfile, const bool _sort) {

```
    size = _size;
    direction_flag = _sort;
    activeLeftHeap = true;
    leftHeapStart = 0;
    leftHeapEnd = 0;
    rightHeapStart = size - 1;
    rightHeapEnd = size;
    dualHeap.resize(size);
```

sort(_infile, _outputfile); }

template <typename t>=""> ReplacementSelectionSort<T>::~ReplacementSelectionSort() {

}

initHeap fills the active heap with elements from the input file template <typename t>=""> T ReplacementSelectionSort<T>::initHeap(istream& infile) {

```
T newElement;


infile >> newElement;

for(int i = 0; i < size && !infile.eof(); i++)
{

    dualHeap[i ]= newElement;
    leftHeapEnd = i;
    heapify(true);
    infile >> newElement;

}
return newElement;
```

}

sort uses initHeap to fill the heap then pops the root from the heap to the current list. New input is added to the active heap or pending heap. Popping add adding input continues until the the input stream is empty. If the pending heap is fills up then the current list is stored and emptied. After the input is exhausted the remaining elements in the heap are popped to the current list. All the lists are then stored into the output file. template <typename t>=""> void ReplacementSelectionSort<T>::sort(std::istream& infile, std::ostream& outputfile) {

```
std::vector<vector<T> >sortedLists;
std::vector<T> currentList;

T newElement;
T lastElement;
bool pendingFull = false;
bool activeHeapOpen;


newElement = initHeap(infile);

while(!infile.eof())
{
```

```
lastElement = pop();


currentList.push_back(lastElement);


if(direction_flag)
{

    if(newElement >= lastElement)
    {

        activeHeapOpen = true;
    }
    else
    {

        activeHeapOpen=false;
    }

}
else
{

    if(newElement <= lastElement)
    {

        activeHeapOpen = true;

    }
    else
    {

        activeHeapOpen = false;
    }
}

if(activeHeapOpen)
{

            current_dualHeap_push(newElement);
}
else
{


    pending_dualHeap_push(newElement);


    if(activeLeftHeap)
    {

        if(leftHeapStart > leftHeapEnd)
        {
            pendingFull = true;
        }

    }
    else
    {

        if(rightHeapStart < rightHeapEnd)
        {
            pendingFull = true;
        }

    }

    if(pendingFull)
    {

        sortedLists.push_back(currentList);
```

```
            currentList.resize(0);

            swapActive();
            pendingFull = false;
        }
    }

    infile >> newElement;

} //END WHILE

if(activeLeftHeap)
{

    while(leftHeapStart <= leftHeapEnd)
    {

        currentList.push_back(pop());
        leftHeapEnd--;

    }
}
else
{


    while(rightHeapStart >= rightHeapEnd)
    {
        currentList.push_back(pop());
        rightHeapEnd++;

    }
}

sortedLists.push_back(currentList);
currentList.resize(0);
swapActive();


if(activeLeftHeap)
{

    while(leftHeapStart <= leftHeapEnd)
    {

        currentList.push_back(pop());
        leftHeapEnd--;

    }
}
else
{

    while(rightHeapStart >= rightHeapEnd)
    {

        currentList.push_back(pop());
        rightHeapEnd++;

    }
}

sortedLists.push_back(currentList);


for(int i = 0; i < sortedLists.size(); i++)
{

    for(int j = 0; j<sortedLists[i].size(); j++)
    {

        outputfile<<sortedLists[i][j]<<" ";
    }
```

```
    outputfile << endl;
}
```

}

heapify moves the right most parent to a leaf if need be template <typename t>=""> void ReplacementSelection↩ Sort<T>::heapify(bool leftSide) {

```
int index;

if(leftSide)
{

    index = parent(leftHeapEnd, leftSide);

    while(index >= leftHeapStart)
    {
        siftDown(index, leftSide);
        index -= 1;
    }

}
else
{

    index = parent(rightHeapEnd,leftSide);

    while(index<=rightHeapStart)
    {

        siftDown(index,leftSide);
        index += 1;
    }
}
```

}

siftUp allows a new element to find a proper position in the heap by going toward the root. template <typename t>=""> void ReplacementSelectionSort<T>::siftUp(int index, bool leftSide) {

```
int p = parent(index, leftSide);

int toSwap = index;

if(leftSide && direction_flag)
{

    if((p >= leftHeapStart) && (dualHeap[index] < dualHeap[p]))
        toSwap=p;
}

else if(leftSide)
{

    if((p >= leftHeapStart) && (dualHeap[index] > dualHeap[p]))
        toSwap = p;
}
else if(direction_flag)
{

    if((p <= rightHeapStart) && (dualHeap[index] < dualHeap[p]))
        toSwap = p;
}
else
{

    if((p <= rightHeapStart) && (dualHeap[index] > dualHeap[p]))
        toSwap = p;
```

```
}


if(toSwap != index)
{

    swap(index,toSwap);
    siftUp(toSwap,leftSide);
}
```

}

siftDown allows a new element to find a proper position in the heap by going toward the leaves. template <typename t>=""> void ReplacementSelectionSort<T>::siftDown(int index, bool leftSide) {

```
int l,
    r,
    toSwap;


l = left(index,leftSide);
r = right(index,leftSide);

if(leftSide && direction_flag)
{

    if(l <= leftHeapEnd && dualHeap[l] < dualHeap[index])
        toSwap = l;
    else
        toSwap = index;

    if(r <= leftHeapEnd && dualHeap[r] < dualHeap[toSwap])
        toSwap = r;
}

else if(leftSide)
{

    if(l <= leftHeapEnd && dualHeap[l] > dualHeap[index])
        toSwap = l;
    else
        toSwap=index;

    if(r <= leftHeapEnd && dualHeap[r] > dualHeap[toSwap])
        toSwap = r;
}
else if(direction_flag)
{

    if(l >= rightHeapEnd && dualHeap[l] < dualHeap[index])
        toSwap = l;
    else
        toSwap = index;

    if(r >= rightHeapEnd && dualHeap[r] < dualHeap[toSwap])
        toSwap = r;
}
else
{

    if(l >= rightHeapEnd && dualHeap[l] > dualHeap[index])
        toSwap = l;
    else
        toSwap = index;

    if(r >= rightHeapEnd && dualHeap[r] > dualHeap[toSwap])
        toSwap = r;
}

if(toSwap != index)
{
```

```
    swap(index,toSwap);
    siftDown(toSwap, leftSide);
}
```

}

pop saves the root of the active heap. Moves the last element of the active heap to the root. Lets the new root settle in the heap. Returns the original root. template <typename t>=""> T ReplacementSelectionSort<T>::pop() {

```
T output;

if(activeLeftHeap)
{

    output = dualHeap[leftHeapStart];
    swap(leftHeapStart, leftHeapEnd);
    leftHeapEnd--;

    siftDown(leftHeapStart, activeLeftHeap);
    leftHeapEnd++;
}
else
{


    output = dualHeap[rightHeapStart];
    swap(rightHeapStart, rightHeapEnd);
    rightHeapEnd++;

    siftDown(rightHeapStart, activeLeftHeap);
    rightHeapEnd--;
}

return output;
```

}

current_dualHeap_push puts an element into the active heap as the last leave then lets it move up the heap template <typename t>=""> void ReplacementSelectionSort<T>::current_dualHeap_push(T entry) {

```
if(activeLeftHeap)
{

    dualHeap[leftHeapEnd] = entry;
    siftUp(leftHeapEnd, activeLeftHeap);

}
else
{

    dualHeap[rightHeapEnd] = entry;
    siftUp(rightHeapEnd, activeLeftHeap);
}
```

}

pending_dualHeap_push puts an element into the pending heap as the last leave then lets it move up the heap template <typename t>=""> void ReplacementSelectionSort<T>::pending_dualHeap_push(T entry) {

```
if(activeLeftHeap)
{
    leftHeapEnd--;
    rightHeapEnd--;
    dualHeap[rightHeapEnd]=entry;
    siftUp(rightHeapEnd,!activeLeftHeap);
}
else
{
    leftHeapEnd++;
    rightHeapEnd++;
    dualHeap[leftHeapEnd]=entry;
    siftUp(leftHeapEnd,!activeLeftHeap);
}
```

}

swapActive swaps the active and pending heaps template <typename t>=""> void ReplacementSelectionSort<↩ T>::swapActive() {

```
activeLeftHeap =! activeLeftHeap;
```

}

swap Swaps the elements at the given indexes template <typename t>=""> void ReplacementSelectionSort<↩ T>::swap(int index1, int index2) {

```
T temp;
temp = dualHeap[index1];
dualHeap[index1] = dualHeap[index2];
dualHeap[index2] = temp;
```

}

left returns the index of the left child template <typename t>=""> int ReplacementSelectionSort<T>::left(int index, bool leftSide) {

```
return (leftSide? 2 * index + 1 : rightHeapStart - (rightHeapStart-index) * 2 - 1);
```

}

Right returns the index of the right child template <typename t>=""> int ReplacementSelectionSort<T>::right(int index, bool leftSide) {

```
return (leftSide? 2 * index + 2 : rightHeapStart - (rightHeapStart - index) * 2 - 2);
```

}

parent returns the index of the parent template <typename t>=""> int ReplacementSelectionSort<T>::parent(int index, bool leftSide) {

```
return (leftSide? (index - 1) / 2 : (index - rightHeapStart + 1) / 2 + rightHeapStart);
```

}

template class ReplacementSelectionSort<int>; template class ReplacementSelectionSort<string>; template class ReplacementSelectionSort<double>; template class ReplacementSelectionSort<float>;

#ifndef TOURNAMENT_SORT #define TOURNAMENT_SORT #include <vector> #include <iostream> #include <fstream> #include <functional> #include <typeinfo> #include <string> #include <queue> #include <algorithm> #include <functional> using namespace std;

template <class t>=""> class TournamentSort { public: int completedRuns; vector<T> finalVector;

helper node class

```
class Node
{
    public:

    int position;
    int rn;
    T value ;

    Node(int x, int y, T z){position = x; rn = y; value = z;}
    virtual ~Node(){}


};


struct compare
{
    bool operator()(const Node& lhs, const Node& rhs)
    {


            return lhs.value < rhs.value;

    }

};

TournamentSort(){}
virtual ~TournamentSort(){}
```

this function initializes the prioroty queue std::priority_queue<Node, std::vector<Node>,compare> initPQ(std↩ ::priority_queue <Node, std::vector<Node>, compare > PQ, std::vector<vector<T> > sortedLists) { completed↩ Runs = 0;

```
    for (int i = 0; i< sortedLists.size(); i++)
    {
        vector<T> run = sortedLists[i];
        T v = run[0];
        Node n(0, i, v);
        PQ.push(n);

    }


    return PQ;

}
```

this function pushes the smallest item in the priority queue and repopulates it void pushToFinal(std::priority_queue <Node, std::vector<Node>, compare > PQ, vector<T> final, std::vector<vector<T> > sortedLists) { std::sort(final.begin(), final.end()); Node smallest = PQ.top();

PQ.pop();

final.push_back(smallest.value);

replace the element popped from PQ if there is another item in that run std::vector<T> run = sortedLists[smallest.↩ rn]; if(smallest.position < run.size()-1) { Node newNode = {(smallest.position + 1), smallest.rn, run[smallest.position + 1]};

PQ.push(newNode);

} else { completedRuns++; } if(completedRuns < sortedLists.size()) {

```
pushToFinal(PQ, final, sortedLists);
```

}

else {

```
finalVector = final;
```

}

```
}
```

};

#endif // TOURNAMENT_SORT

4.22 temp.txt File Reference

4.23 TournamentSort.h File Reference

A class to apply tournament sort on a series of sorted lists.

```
#include <vector>
#include <iostream>
#include <fstream>
#include <functional>
#include <typeinfo>
#include <string>
#include <queue>
#include <algorithm>
```

Include dependency graph for TournamentSort.h: This graph shows which files directly or indirectly include this file:

Classes

- class TournamentSort< T >
- class TournamentSort< T >::Node
- struct TournamentSort< T >::compare

4.23.1 Detailed Description

A class to apply tournament sort on a series of sorted lists.

4.24 UserManual.cpp File Reference

4.24.1 Detailed Description

Replacement Selection Sort User Manual CSCI 331 Team 3 Program name: ReplacementSelectionSort and TournamentSort. What is the program supposed to do?: The program will take files of data, keys, and sort them in order depending on if the keys are strings of characters or a string of integers. The program will then go through and put the output into its own file. In this file some other data is given like the number of runs the program had and the average run length in the program. The Tournament sort will go through and merge the files into one file of records. location of the program: :/export/home/cs331/cs331128/Desktop/331/finalProduct flags: -s -a String ascending -s String descending -i -a Integer ascending -i Integer descending -d -a Double ascending -d Double descending -f -a Float ascending -f Float descending -a Run all tests sequentially makefile:

mainProgram: mainProgram.cpp ReplacementSelectionSort.cpp ReplacementSelectionSort.h TournamentSort.↩ h
 -o mainProgram mainProgram.o ReplacementSelectionSort.o TournamentSort.o
 test run:

AB AB AZ AZ Ar Bh CE CY DW Da EA Eb FD Gb Gb Gh Gs HD Hu IM JI JL JW MM MT NM Pa Pf Pi QR VZ Vt Vy Wi Yq aU bW bv cB cY dK dT dy eG eS gn iq jE ji kp lS mj nF ot ra rb rb sK tX tq vR wv xx zi zv

End of test string Ascending = true sE ps pc pK oj oj he cI ar Zq ZR Yh Ui Te Mf KH KE Jd Ex BG AA

End of test string Ascending = false 10 10 10 10 10 11 11 11 11 13 13 13 13 13 14 15 15 16 17 19 19 22 22 24 25 25 25 25 25 25 26 26 26 26 27 27 27 27 27 28 28 29 29 29 32 32 32 33 33 33 33 35 35 35 36 36 36 37 37 37 39 42 42 43 44 44 44 45 45 46 46 46 47 47 47 48 48 48 49 49 50 51 52 53 53 53 54 54 55 55 56 56 57 58 58 59 59 60 60 61 61 61 61 62 62 62 63 64 65 66 67 67 67 67 68 69 69 70 70 71 71 72 72 72 72 74 74 74 75 75 75 75 75 76 76 76 77 78 79 79 79 79 79 80 80 80 80 81 81 82 82 83 84 84 85 85 85 85 87 87 87 87 87 87 88 88 89 89 89 90 91 91 92 93 93 93 94 95 95 97 98 98 98

End of test integersAscending = 1 98 98 98 98 98 98 98 98 98 98 97 97 97 97 97 97 97 97 97 97 97 97 96 96 96 96 96 96 96 96 96 96 96 96 95 95 95 95 95 95 95 95 95 95 94 94 94 94 94 94 94 94 93 93 93 93 93 93 93 93 93 92 92 92 92 92 92 92 92 92 92 91 91 91 91 91 91 91 91 91 91 91 90 90 90 90 90 90 90 90 90 90 89 89 89 89 89 89 88 88 88 88 87 87 87 87 87 87 87 87 87 87 86 86 86 86 86 86 86 86 86 86 86 85 85 85 85 85 85 85 85 85 85 84 84 84 84 84 84 84 84 83 83 83 83 83 83 83 83 83 83 83 83 82 82 82 82 82 82 82 82 82 82 82 81 81 81 81 81 81 81 81 81 81 80 80 80 80 80 80 80 80 80 79 79 79 79 79 79 79 79 79 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 76 76 76 76 76 76 75 75 75 75 75 75 74 74 74 74 74 74 74 74 73 73 73 73 73 73 73 73 73 73 73 73 72 72 72 72 72 72 72 71 71 71 71 71 71 71 71 70 70 70 70 70 70 70 69 69 69 69 69 69 69 69 69 69 69 69 68 68 68 68 68 68 67 67 67 67 67 67 66 66 66 66 66 66 66 65 65 65 65 65 65 65 65 65 65 65 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 63 63 63 63 63 63 63 63 63 63 63 62 62 62 62 62 62 62 62 62 62 62 62 62 61 61 61 61 61 61 61 61 60 60 60 60 60 60 59 59 59 59 59 58 58 58 58 58 58 58 58 58 58 58 58 58 58 57 57 57 57 57 57 57 57 56 56 56 56 56 56 56 55 55 55 55 55 55 55 54 54 54 54 54 54 54 54 54 53 53 53 53 53 53 53 52 52 52 52 52 52 52 52 52 52 52 52 52 52 51 51 51 51 51 51 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 49 49 49 49 49 49 49 48 48 48 48 48 48 48 48 47 47 47 47 47 47 46 46 46 46 46 46 46 45 45 45 45 45 44 44 44 44 44 44 44 43 43 43 43 43 43 43 43 43 43 43 43 43 43 42 42 42 42 42 42 42 42 42 42 42 42 42 41 41 41 41 41 41 41 41 41 41 41 40 40 40 40 40 39 39 39 39 39 39 39 39 39 38 38 38 38 38 38 38 38 38 38 38 38 38 37 37 37 37 37 37 37 37 37 37 37 37 37 37 36 36 36 36 36 36 36 35 35 35 35 35 35 35 35 34 34 34 34 34 34 34 34 34 34 34 34 34 33 33 33 33 33 33 33 33 33 33 33 33 33 32 32 32 32 32 32 32 32 32 32 31 31 31 31 31 31 31 31 30 30 30 30 30 30 30 30 30 30 30 29 29 29 29 29 29 28 28 28 28 28 28 28 28 28 27 27 27 27 27 27 27 27 26 26 26 26 26 26 26 26 26 26 26 25 25 25 25 25 25 25 25 25 25 25 25 24 24 24 24 24 24 23 23 23 23 23 23 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 21 21 21 21 21 21 21 21 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 19 19 19 19 19 19 19 19 19 19 19 19 18 18 18 18 18 18 17 17 17 17 17 17 17 17 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 15 15 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13 13 13 13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

End of test integersAscending = 0 10 10 10 10 11 11 12 12 12 12 12 12 12 12 13 13 13 14 14 14 14 14 14 14 14 15 15 15 15 15 15 16 16 16 16 17 17 18 18 18 18 19 19 19 20 20 20 21 21 21 21 21 22 23 23 23 23 24 24 24 24

25 25 25 25 25 26 26 26 27 27 28 28 28 28 28 29 29 29 30 31 31 31 32 32 32 32 33 33 33 33 33 34 34 34 35 35
35 36 36 36 37 38 38 39 40 40 40 41 41 42 42 43 43 43 43 43 43 43 43 44 44 44 45 45 46 46 47 47 47 47 48 48
48 48 48 49 49 49 49 50 50 51 51 51 51 51 51 52 52 52 53 53 53 53 53 53 54 54 54 55 55 55 55 56 56 57 57
57 57 58 58 59 59 59 59 59 59 60 61 61 62 62 62 62 63 63 63 63 64 64 64 65 65 66 66 66 66 66 67 67 67 68 68
68 69 69 69 70 70 71 72 72 72 72 73 73 74 74 74 75 75 75 75 76 76 76 76 76 77 77 77 77 77 77 77 77 78 78 78
79 79 80 80 80 80 82 82 82 82 83 83 83 84 85 85 85 85 86 86 86 86 87 87 87 87 87 87 88 88 88 88 89 89 90 90
91 91 91 91 92 92 92 92 93 93 93 93 94 94 94 94 94 94 95 95 95 95 95 95 97 97 98 98

End of test floatsAscending = 1 98 98 97 97 97 96 96 96 96 95 94 94 93 93 92 92 91 91 90 89 89 89 89 89 89 89
89 88 88 87 87 87 87 87 87 85 85 84 84 82 82 82 82 82 81 81 81 80 79 79 79 78 78 78 77 75 75 74 74 73 72 71
71 71 71 71 71 71 71 70 70 70 69 68 68 68 68 68 67 67 66 66 66 65 65 65 64 64 64 63 63 63 62 62 62 60 60 59
59 59 58 58 58 57 57 56 56 55 54 54 53 53 53 53 53 52 52 51 51 51 51 51 50 48 48 44 44 44 44 43 42 42 42 40
40 40 40 40 39 37 37 37 37 37 37 37 34 34 34 34 33 33 33 33 33 31 30 28 27 27 27 27 26 26 26 25 25 25 25 25
24 24 24 23 23 23 23 22 22 22 22 22 22 21 21 21 21 20 19 19 18 18 18 18 17 17 16 16 16 16 15 15 15 14 14 13
13 12 12 12 12 12 12 12 11 11 11 11 11 10 10 10

End of test floatsAscending = 0 10 10 10 10 10 10 11 11 12 12 12 12 12 12 13 13 13 13 13 13 14 14 14 14 14 14
16 16 16 17 17 17 17 17 17 18 18 18 18 18 18 19 19 20 21 21 21 22 22 23 23 23 24 24 24 26 27 27 28 28 28 28
28 28 29 30 30 30 31 31 31 31 31 31 31 31 32 32 32 32 32 32 33 34 34 35 36 36 37 38 38 38 38 40 40 41 41 41
41 41 42 42 43 44 44 44 45 45 45 46 46 46 46 46 46 47 47 47 48 48 48 48 48 48 49 49 50 50 50 50 51 52 52 52
53 54 55 55 55 55 55 55 56 56 56 57 57 57 59 59 60 60 60 61 62 62 63 63 64 65 65 65 66 66 66 66 66 67 67 67
68 69 71 71 71 73 73 73 73 73 74 74 74 75 76 76 77 77 77 78 78 79 79 80 80 81 82 83 83 83 84 84 84 85 85 86
86 87 87 88 88 89 90 91 91 92 92 93 93 95 95 95 96 96 97 98 98 98

End of test doublesAscending = 1 98 98 98 98 98 98 98 98 98 97 97 97 97 97 97 97 96 96 96 96 96 95 95 95 94
94 94 94 93 93 93 93 93 93 93 92 92 92 92 92 92 92 92 92 91 91 91 91 91 90 90 90 90 90 90 89 89 89 89 89 89
88 88 87 87 87 87 87 87 87 87 86 86 86 86 86 85 85 85 85 85 84 84 84 83 83 83 83 83 82 82 82 82 82 82 81 81
81 81 81 81 81 81 80 80 80 80 80 80 80 79 79 79 79 79 78 78 78 77 77 77 77 77 77 77 77 77 77 77 76 76 76 76
76 76 75 75 75 75 75 74 74 73 73 73 73 73 73 72 72 72 72 71 71 71 70 70 69 69 69 69 68 68 68 68 68 67 67 67
67 67 67 67 66 66 66 66 66 66 66 65 65 65 65 65 64 64 64 64 64 63 63 63 63 63 63 62 62 62 62 62 62 62 62 61
61 61 60 60 60 59 59 59 59 59 59 58 58 58 57 57 57 57 57 57 57 57 56 56 56 56 56 56 56 55 55 55 55 55 54
54 54 53 53 53 53 53 52 52 52 52 52 52 52 52 52 51 51 51 51 51 51 51 51 50 50 50 49 49 49 49 49 48 48 48 48
48 48 48 48 48 48 47 47 47 47 47 47 47 47 46 46 46 46 46 46 46 46 45 45 45 45 45 44 44 44 43 43 43 43 43 42
42 41 41 41 41 41 40 40 40 40 39 39 39 39 39 38 38 38 37 37 37 37 36 36 36 36 36 35 35 35 35 35 35 35 35 35
34 34 33 33 33 33 33 33 33 33 33 33 33 33 32 32 32 32 32 31 31 31 31 31 31 31 31 31 31 30 30 30 30 30 30 29
29 29 29 29 28 28 28 28 28 28 28 27 27 27 27 27 27 27 26 26 26 25 25 25 25 25 25 25 25 25 24 24 24 24 24 24
24 23 22 22 22 22 22 21 21 21 20 20 20 20 20 20 19 19 19 19 18 18 18 18 18 18 18 18 18 17 17 17 17 17 16 16
15 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 13 13 13 13 13 13 13 13 13 12 12 12 12 12 12 12 12 12 12
12 12 12 12 11 10 10 10 10 10 10 10 10 10 10 10 10

End of test doublesAscending = 0

Index