Data Structures - 20407 - Maman 13

Generated by Doxygen 1.9.1

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Class Documentation

4.1 Game Class Reference

Public Member Functions

- void **run** ()

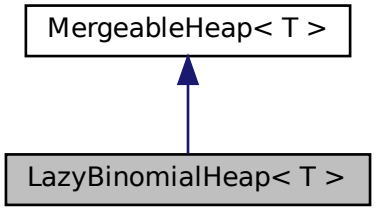
The documentation for this class was generated from the following file:

- game.hpp

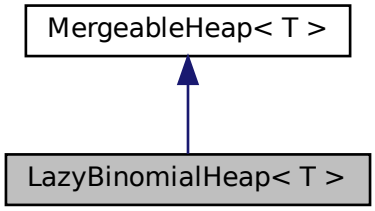
4.2 LazyBinomialHeap< T > Class Template Reference

A fast mergeable heap data structure implementation.

Inheritance diagram for LazyBinomialHeap< T >:



Collaboration diagram for LazyBinomialHeap< T >:



Public Member Functions

- constexpr LazyBinomialHeap ()

  Constructs a new empty heap.
- constexpr ~LazyBinomialHeap ()

  Destroys the heap.
- constexpr void insert (T key) override

  Inserts a key into the heap.
- constexpr std::optional< std::reference_wrapper< const T > > minimum () const override

  Returns the minimum key in the heap.
- constexpr std::optional< T > extract_min () override

  Removes and returns the minimum key in the heap.
- constexpr void merge (MergeableHeap< T > &other) override

  Merges another heap into this heap.
- void print () const

  Prints the heap.
- void sort () override

  Sorts the heap in ascending order.

Additional Inherited Members

4.2.1 Detailed Description

template<typename T>
class LazyBinomialHeap< T >

This mergeable heap is implemented using the binomial heap data structure. A binomial heap is a collection of binomial trees, where each binomial tree is a heap-ordered tree. A binomial tree of order k is a tree with 2^k nodes, where each node has a key and a pointer to its parent and its leftmost child. The binomial heap is a collection of binomial trees of different orders, where each order is represented by a linked list of binomial trees. The root list of the heap is a linked list of the roots of the binomial trees in the heap, hence its size is logarithmic in the number of nodes in the heap in the average case. The current implementation of the binomial heap is lazy, meaning that the heap is consolidated only when needed, during the extract_min operation. This allows for O(1) insertions and merges, and an amortized O(log n) extract_min. Amortized analysis is used to analyze the time complexity of the consolidate operation,

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 LazyBinomialHeap() `template<typename T >`

`constexpr LazyBinomialHeap< T >::LazyBinomialHeap ( ) [inline], [constexpr]`

This constructor initializes the heap to be empty. The head, tail, and minimum node pointers are all set to `nullptr`, and the size is set to 0.

The time complexity of this operation is O(1).

4.2.2.2 ~LazyBinomialHeap() `template<typename T >`

`constexpr LazyBinomialHeap< T >::~LazyBinomialHeap ( ) [inline], [constexpr]`

This destructor deletes the head node of the heap, which recursively deletes all nodes in the heap.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

4.2.3 Member Function Documentation

4.2.3.1 extract_min()

```
template<typename T >
constexpr std::optional<T> LazyBinomialHeap< T >::extract_min ( ) [inline], [constexpr],
[override], [virtual]
```

This method removes the minimum key from the heap and returns it. If the minimum key has children, they are added to the root list. The heap is then consolidated to maintain the heap property. If the heap is empty, `std::nullopt` is returned.

The time complexity of this operation is O(log n) in the average case (amotrized), and O(n) in the worst case where n is the number of nodes in the heap. This is because the heap is consolidated after the minimum key is removed, which involves linking binomial trees of the same degree until there are no two trees with the same degree.

Returns

The minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.2.3.2 insert()

```
template<typename T >
constexpr void LazyBinomialHeap< T >::insert (
            T key ) [inline], [constexpr], [override], [virtual]
```

This method inserts a new key into the heap and updates the minimum accordingly.

The time complexity of this operation is O(1), because it only involves creating a new node and updating the head and minimum pointers. The heap is not consolidated after each insertion, so the heap may become unbalanced. This is fine because the heap is consolidated lazily only when needed, during the extract_min operation.

Parameters

key	The key to insert. This key is moved into the heap.

Implements MergeableHeap< T >.

4.2.3.3 merge()

```
template<typename T >
constexpr void LazyBinomialHeap< T >::merge (
            MergeableHeap< T > & other ) [inline], [constexpr], [override], [virtual]
```

This method merges another heap into this heap. The root list of the other heap is concatenated to the root list of this heap, and the size of this heap is increased by the size of the other heap, and the minimum is updated accordingly.

Note

The other heap is left empty after the merge.

The time complexity of this operation is O(1), because it only involves updating the head, tail, minimum and size pointers of this heap. The heap is not consolidated after the merge, so the heap may become unbalanced. This is fine because the heap is consolidated lazily only when needed, during the extract_min operation.

Parameters

other	The heap to merge into this heap.

Implements MergeableHeap< T >.

4.2.3.4 minimum() `template<typename T >`

`constexpr std::optional<std::reference_wrapper<const T> > LazyBinomialHeap< T >::minimum ( ) const [inline], [constexpr], [override], [virtual]`

This method returns a reference to the minimum key in the heap, or `std::nullopt` if the heap is empty. The reference is wrapped in a `std::reference_wrapper` to allow it to be used in contexts where a reference cannot be used.

The time complexity of this operation is O(1), because the minimum key is stored in the `min` pointer, which is updated whenever a new minimum key is inserted.

Returns

A reference to the minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.2.3.5 print() `template<typename T >`

`void LazyBinomialHeap< T >::print ( ) const [inline], [virtual]`

This method prints the keys of the nodes in the heap using a breadth-first traversal.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

Implements MergeableHeap< T >.

4.2.3.6 sort() `template<typename T >`

`void LazyBinomialHeap< T >::sort ( ) [inline], [override], [virtual]`

This method extracts the minimum key from the heap until the heap is empty. The extracted keys are then inserted into a temporary heap, which is then merged back into the original heap.

The time complexity of this operation is O(n log n), where n is the number of nodes in the heap. This is because the heap is sorted by extracting the minimum key n times, each of which takes O(log n) time, resulting in a total time complexity of O(n log n).

Implements MergeableHeap< T >.

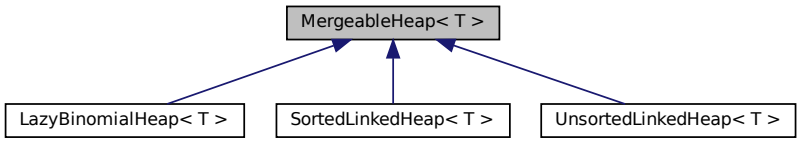
The documentation for this class was generated from the following file:

- lazy.h

4.3 MergeableHeap< T > Class Template Reference

A constexpr-friendly interface for a mergeable heap data structure.

Inheritance diagram for MergeableHeap< T >:



Public Member Functions

- constexpr MergeableHeap ()=default
- constexpr virtual ~MergeableHeap ()=default
- constexpr MergeableHeap (const MergeableHeap &)=delete

  Deleted special member functions.
- constexpr MergeableHeap & **operator=** (const MergeableHeap &)=delete
- constexpr **MergeableHeap** (MergeableHeap &&)=delete
- constexpr MergeableHeap & **operator=** (MergeableHeap &&)=delete
- constexpr virtual void insert (T key)=0
- constexpr virtual std::optional< std::reference_wrapper< const T > > minimum () const =0
- constexpr virtual std::optional< T > extract_min ()=0
- constexpr virtual void merge (MergeableHeap< T > &other)=0
- virtual void print () const =0
- virtual void sort ()=0

  Sorts the heap in ascending order.

Protected Member Functions

- void sort (MergeableHeap< T > &&temp)

  Sorts the heap using a temporary heap.

4.3.1 Detailed Description

template<typename T>
class MergeableHeap< T >

A mergeable heap is a type of heap that supports the following operations:

1. MAKE-HEAP creates a new (empty) heap.
2. INSERT inserts a new element into the heap.
3. MINIMUM returns the element with the minimum key in the heap.
4. EXTRACT-MIN removes and returns the element with the minimum key in the heap.
5. UNION merges two heaps into a single heap.

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 MergeableHeap() [1/2]

```
template<typename T >
constexpr MergeableHeap< T >::MergeableHeap ( ) [constexpr], [default]
```

Constructs a new empty heap.

4.3.2.2 ~MergeableHeap()

```
template<typename T >
constexpr virtual MergeableHeap< T >::~MergeableHeap ( ) [constexpr], [virtual], [default]
```

Destroys the heap.

4.3.2.3 MergeableHeap() [2/2]

```
template<typename T >
constexpr MergeableHeap< T >::MergeableHeap (
            const MergeableHeap< T > & ) [constexpr], [delete]
```

`MergeableHeap` objects should not be copied or moved.

4.3.3 Member Function Documentation

4.3.3.1 extract_min()

```
template<typename T >
constexpr virtual std::optional<T> MergeableHeap< T >::extract_min ( ) [constexpr], [pure
virtual]
```

Removes and returns the minimum key in the heap.

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.2 insert()

```
template<typename T >
constexpr virtual void MergeableHeap< T >::insert (
            T key ) [constexpr], [pure virtual]
```

Inserts a key into the heap.

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.3 merge()

```
template<typename T >
constexpr virtual void MergeableHeap< T >::merge (
            MergeableHeap< T > & other ) [constexpr], [pure virtual]
```

Merges another heap into this heap.

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.4 minimum()

```
template<typename T >
constexpr virtual std::optional<std::reference_wrapper<const T> > MergeableHeap< T >::minimum
( ) const [constexpr], [pure virtual]
```

Returns the minimum key in the heap.

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.5 print()

```
template<typename T >
virtual void MergeableHeap< T >::print ( ) const [pure virtual]
```

Prints the heap.

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.6 sort() [1/2]

```
template<typename T >
virtual void MergeableHeap< T >::sort ( ) [pure virtual]
```

This function extracts the minimum key from the heap and prints it until the heap is empty. The extracted keys are then inserted into a temporary heap, which is then merged back into the original heap.

Complexity depends on the implementation of the heap: O(extract_min) $*$ O(insert) + O(merge).

Implemented in UnsortedLinkedHeap< T >, SortedLinkedHeap< T >, and LazyBinomialHeap< T >.

4.3.3.7 sort() [2/2]

```
template<typename T >
void MergeableHeap< T >::sort (
            MergeableHeap< T > && temp ) [inline], [protected]
```

This function should be called by the `sort` function of the derived class in the following way: `this->MergeableHeap<T>↩ ::sort(DerivedHeap<T>{})`.

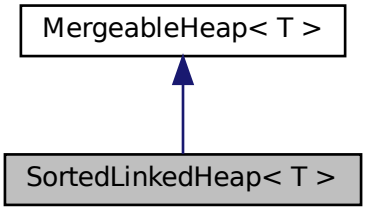
The documentation for this class was generated from the following file:

- mergeable_heap.h

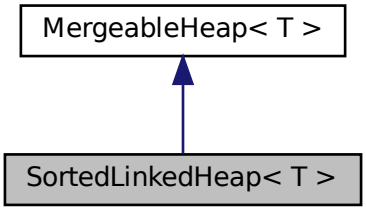
4.4 SortedLinkedHeap< T > Class Template Reference

A mergeable heap data structure implementation using a sorted linked list.

Inheritance diagram for SortedLinkedHeap< T >:



Collaboration diagram for SortedLinkedHeap< T >:



Public Member Functions

- constexpr SortedLinkedHeap ()

  Constructs a new empty heap.
- constexpr ~SortedLinkedHeap ()

  Destroys the heap.
- constexpr void insert (T key) override

  Inserts a key into the heap.
- constexpr std::optional< std::reference_wrapper< const T > > minimum () const override

  Returns the minimum key in the heap.
- constexpr std::optional< T > extract_min () override

  Removes and returns the minimum key in the heap.
- constexpr void merge (MergeableHeap< T > &other) override

  Merges another heap into this heap.
- void print () const override

  Prints the heap.
- void sort () override

  Sorts the heap in ascending order.

Additional Inherited Members

4.4.1 Detailed Description

template<typename T>
class SortedLinkedHeap< T >

This mergeable heap is implemented using a sorted linked list. The heap is maintained in a sorted order, where the minimum key is always at the head of the list.

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SortedLinkedHeap() `template<typename T >`

`constexpr SortedLinkedHeap< T >::SortedLinkedHeap ( ) [inline], [constexpr]`

This constructor initializes the heap to be empty. The head is set to `nullptr`.

The time complexity of this operation is O(1).

4.4.2.2 ~SortedLinkedHeap() `template<typename T >`

`constexpr SortedLinkedHeap< T >::~SortedLinkedHeap ( ) [inline], [constexpr]`

The destructor deletes all nodes in the linked list.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

4.4.3 Member Function Documentation

4.4.3.1 extract_min() `template<typename T >`

`constexpr std::optional<T> SortedLinkedHeap< T >::extract_min ( ) [inline], [constexpr], [override], [virtual]`

This method removes the minimum key from the heap and returns it. If the heap is empty, the method returns `std::nullopt`.

The time complexity of this operation is O(1), because the minimum key is stored in the `head` pointer, which is always available at constant time.

Returns

The minimum key in the heap, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.4.3.2 insert()

```
template<typename T >
constexpr void SortedLinkedHeap< T >::insert (
            T key )  [inline], [constexpr], [override], [virtual]
```

This function inserts a new key into the heap. The key is inserted at the correct position in the linked list to maintain the sorted order of the heap.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

Parameters

key	The key to insert into the heap. This key is moved into the heap.

Implements MergeableHeap< T >.

4.4.3.3 merge()

```
template<typename T >
constexpr void SortedLinkedHeap< T >::merge (
            MergeableHeap< T > & other )  [inline], [constexpr], [override], [virtual]
```

This method merges another heap into this heap. The other heap is emptied in the process, and its nodes are inserted into this heap in sorted order.

Note

The other heap is left empty after the merge.

The time complexity of this operation is O(n + m), where n is the number of nodes in this heap and m is the number of nodes in the other heap.

Parameters

other	The heap to merge into this heap.

Implements MergeableHeap< T >.

4.4.3.4 minimum()

```
template<typename T >
constexpr std::optional<std::reference_wrapper<const T> > SortedLinkedHeap< T >::minimum ( )
const  [inline], [constexpr], [override], [virtual]
```

This method returns a reference to the minimum key in the heap, or `std::nullopt` if the heap is empty. The reference is wrapped in a `std::reference_wrapper` to allow it to be used in contexts where a reference cannot be used.

The time complexity of this operation is O(1), because the minimum key is stored in the `head` pointer, which is always available at constant time.

Returns

A reference to the minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.4.3.5 print() `template<typename T >`

`void SortedLinkedHeap< T >::print ( ) const [inline], [override], [virtual]`

This function prints the keys in the heap in sorted order. The keys are separated by commas and followed by a period.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

Implements MergeableHeap< T >.

4.4.3.6 sort() `template<typename T >`

`void SortedLinkedHeap< T >::sort ( ) [inline], [override], [virtual]`

This method extracts the minimum key from the heap and until the heap is empty. The extracted keys are then inserted into a temporary heap, which is then merged back into the original heap.

The time complexity of this operation is O(n log n), where n is the number of nodes in the heap. This is because the heap is sorted by extracting the minimum key n times, each of which takes O(log n) time, resulting in a total time complexity of O(n log n).

Note

The function could be O(1) because the heap is already sorted, but the requirement is to extract the minimum key n times, which is O(n log n).

Implements MergeableHeap< T >.

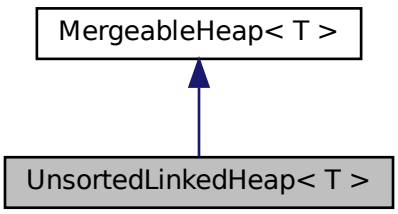
The documentation for this class was generated from the following file:

- sorted.h

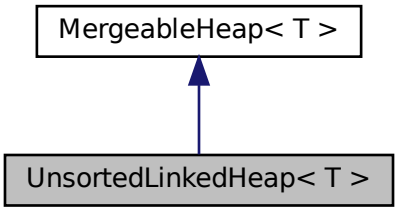
4.5 UnsortedLinkedHeap< T > Class Template Reference

A mergeable heap data structure implementation using an unsorted linked list.

Inheritance diagram for UnsortedLinkedHeap< T >:



Collaboration diagram for UnsortedLinkedHeap< T >:



Public Member Functions

- constexpr UnsortedLinkedHeap ()

  Constructs a new empty heap.
- constexpr ~UnsortedLinkedHeap ()

  Destroys the heap.
- constexpr void insert (T key) override

  Inserts a key into the heap.
- constexpr std::optional< std::reference_wrapper< const T > > minimum () const override

  Returns the minimum key in the heap.
- constexpr std::optional< T > extract_min () override

  Removes and returns the minimum key in the heap.
- constexpr void merge (MergeableHeap< T > &other) override

  Merges another heap into this heap.
- void print () const override

  Prints the heap.
- void sort () override

  Sorts the heap in ascending order.

Additional Inherited Members

4.5.1 Detailed Description

template<typename T>
class UnsortedLinkedHeap< T >

This mergeable heap is implemented using an unsorted linked list. The heap is maintained in an unsorted order, with pointers to the tail and the minimum nodes in the list to allow for constant time insertion, merge and extraction of the minimum key.

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 UnsortedLinkedHeap()

```
template<typename T >
constexpr UnsortedLinkedHeap< T >::UnsortedLinkedHeap ( ) [inline], [constexpr]
```

This constructor initializes the heap to be empty. The head, tail and min are set to `nullptr`.

The time complexity of this operation is O(1).

4.5.2.2 ~UnsortedLinkedHeap()

```
template<typename T >
constexpr UnsortedLinkedHeap< T >::~UnsortedLinkedHeap ( ) [inline], [constexpr]
```

The destructor deletes all nodes in the linked list.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

4.5.3 Member Function Documentation

4.5.3.1 extract_min()

```
template<typename T >
constexpr std::optional<T> UnsortedLinkedHeap< T >::extract_min ( ) [inline], [constexpr],
[override], [virtual]
```

This method removes the minimum key from the heap and returns it. If the heap is empty, the method returns `std::nullopt`.

The time complexity of this operation is O(n), where n is the number of nodes in the heap, because the new minimum key must be searched for in the entire linked list.

Returns

The minimum key in the heap, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.5.3.2 insert()

```
template<typename T >
constexpr void UnsortedLinkedHeap< T >::insert (
            T key ) [inline], [constexpr], [override], [virtual]
```

This function inserts a new key into the heap. The key is inserted at the end of the linked list, and the minimum key is updated if the new key is smaller than the current minimum.

The time complexity of this operation is O(1).

Parameters

key	The key to insert into the heap. This key is moved into the heap.

Implements MergeableHeap< T >.

4.5.3.3 merge()

```
template<typename T >
constexpr void UnsortedLinkedHeap< T >::merge (
            MergeableHeap< T > & other )  [inline], [constexpr], [override], [virtual]
```

This function merges another heap into this heap. The other heap is emptied in the process, and its nodes are moved into this heap. The minimum key is updated if the new heap contains a smaller key than the current minimum.

Note

The other heap is left empty after the merge.

The time complexity of this operation is O(1), because the other heap is merged into this heap by concatenating the tail of the other heap with the head of this heap.

Parameters

other	The heap to merge into this heap.

Implements MergeableHeap< T >.

4.5.3.4 minimum()

```
template<typename T >
constexpr std::optional<std::reference_wrapper<const T> > UnsortedLinkedHeap< T >::minimum (
) const  [inline], [constexpr], [override], [virtual]
```

This method returns a reference to the minimum key in the heap, or `std::nullopt` if the heap is empty. The reference is wrapped in a `std::reference_wrapper` to allow it to be used in contexts where a reference cannot be used.

The time complexity of this operation is O(1), because the minimum key is stored in the `min` pointer, which is always available at constant time.

Returns

A reference to the minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

4.5.3.5 print() template<typename T >

```
void UnsortedLinkedHeap< T >::print ( ) const [inline], [override], [virtual]
```

This function prints the keys in the heap in the order they are stored in the linked list. The keys are separated by commas and followed by a period.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

Implements MergeableHeap< T >.

4.5.3.6 sort() template<typename T >

```
void UnsortedLinkedHeap< T >::sort ( ) [inline], [override], [virtual]
```

This method extracts the minimum key from the heap until the heap is empty. The extracted keys are then inserted into a temporary heap, which is then merged back into the original heap.

The time complexity of this operation is O(n^2), where n is the number of nodes in the heap, because the minimum key must be extracted n times, and each extraction takes O(n) time.

Implements MergeableHeap< T >.

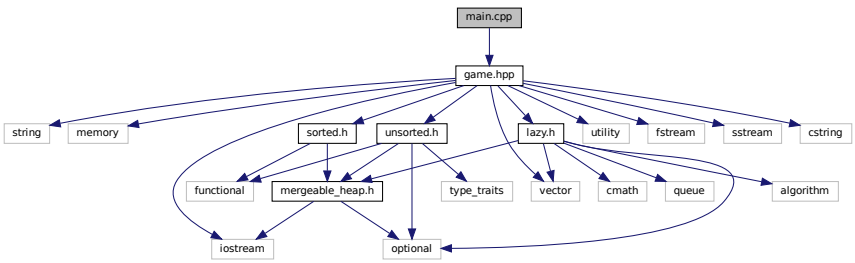
The documentation for this class was generated from the following file:

- unsorted.h

5 File Documentation

5.1 main.cpp File Reference

Include dependency graph for main.cpp:



Functions

- int **main** ()

5.1.1 Detailed Description

Author

Yehonatan Simian

Date

June 2024

```
+-------------------------------------------------------------------------------+
|                        20407 - Data Structures - Maman 13                     |
|                                                                               |
|                 "One must imagine Sisyphus happy." - Albert Camus             |
+-------------------------------------------------------------------------------+
```

5.1.2 DESCRIPTION

This project includes three implementations of a mergeable heap data structure:

1. UnsortedLinkedHeap: A mergeable heap implemented using an unsorted linked list.
2. SortedLinkedHeap: A mergeable heap implemented using a sorted linked list.
3. LazyHeap: A mergeable heap implemented using a lazy binomial heap.

The project also includes a Game class that allows the user to interact with the heaps through a command-line interface. The user can create, insert, extract, merge, print, and sort heaps.

Complexity table:

Operation	UnsortedLinkedHeap	SortedLinkedHeap	LazyBinomialHeap
MAKE-HEAP	O(1)	O(1)	O(1)
INSERT	O(1)	O(n)	O(1)
MINIMUM	O(1)	O(1)	O(1)
EXTRACT-MIN	O(n)	O(1)	O(log n) amortized
UNION	O(1)	O(n+m)	O(1)

Note

My implementations of the mergeable heap are all constexpr-friendly, but do not handle runtime memory allocation failures for the sake of simplicity.

5.1.3 USAGE

The program is very self explanatory. Better to run it and see for yourself!

5.1.4 COMPILATION

To compile this project, I have used the following command:

g++ -std=c++23 -Wall -Wextra -Werror -Wpedantic -o main main.cpp

Note

My implementations use C++23, and were tested using gcc 13.1.0.

Copyright

Index