CS121 Data Structures A, C
Priority Queue Sorting

Varduhi Yeghiazaryan
vyeghiazaryan@aua.am

Fall 2021

Sorting Scheme with a Priority Queue

We can use a priority queue to sort a list of comparable elements

1. insert the elements one by one with a series of `insert` operations
2. remove the elements in sorted order with a series of `removeMin` operations

The running time of this sorting method depends on the priority queue implementation

```
1  /** Sorts sequence S, using initially empty priority queue P to produce the order. */
2  public static <E> void pqSort(PositionalList<E> S, PriorityQueue<E,?> P) {
3    int n = S.size( );
4    for (int j=0; j < n; j++) {
5      E element = S.remove(S.first( ));
6      P.insert(element, null);          // element is key; null value
7    }
8    for (int j=0; j < n; j++) {
9      E element = P.removeMin( ).getKey( );
10     S.addLast(element);               // the smallest key in P is next placed in S
11   }
12 }
```

Selection Sort

Selection sort is the variation of PQ-sort where the priority queue is implemented with an unsorted sequence

Running time of Selection sort:

1. Inserting the elements into the priority queue with n `insert` operations takes $O(n)$ time
2. Removing the elements in sorted order from the priority queue with n `removeMin` operations takes time proportional to

$$n + (n-1) + \cdots + 1$$

Selection sort runs in $O(n^2)$ time

Selection Sort Example

	Sequence S	Priority Queue P
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(7,4)
$\vdots$	$\vdots$	$\vdots$
(g)	()	(7,4,8,2,5,3,9)
Phase 2		
(a)	(2)	(7,4,8,5,3,9)
(b)	(2,3)	(7,4,8,5,9)
(c)	(2,3,4)	(7,8,5,9)
(d)	(2,3,4,5)	(7,8,9)
(e)	(2,3,4,5,7)	(8,9)
(f)	(2,3,4,5,7,8)	(9)
(g)	(2,3,4,5,7,8,9)	()

Insertion Sort

Insertion sort is the variation of PQ-sort where the priority queue is implemented with a sorted sequence

Running time of Insertion sort:

1. Inserting the elements into the priority queue with n `insert` operations takes time proportional to

$$1 + 2 + \cdots + n$$

2. Removing the elements in sorted order from the priority queue with a series of n `removeMin` operations takes $O(n)$ time

Insertion sort runs in $O(n^2)$ time

Insertion Sort Example

	Sequence S	Priority Queue P
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(4,7)
(c)	(2,5,3,9)	(4,7,8)
(d)	(5,3,9)	(2,4,7,8)
(e)	(3,9)	(2,4,5,7,8)
(f)	(9)	(2,3,4,5,7,8)
(g)	()	(2,3,4,5,7,8,9)
Phase 2		
(a)	(2)	(3,4,5,7,8,9)
(b)	(2,3)	(4,5,7,8,9)
⋮	⋮	⋮
(g)	(2,3,4,5,7,8,9)	()

Heap Sort

Consider a priority queue with n items implemented by means of a heap

- the space used is $O(n)$
- methods `insert` and `removeMin` take $O(\log n)$ time
- methods `size`, `isEmpty`, and `min` take $O(1)$ time

Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time

The resulting algorithm is called **heap sort**

Heap sort is much faster than quadratic sorting algorithms, such as insertion sort and selection sort

Summary

Reading

Section 9.4 Sorting with a Priority Queue

Questions?