CPSC 531: System Modeling and Simulation

Carey Williamson
Department of Computer Science
University of Calgary
Fall 2017

UNIVERSITY OF CALGARY

Event List Management

- Event List:
  - Data structure containing the events that are scheduled to occur in the future in the simulation
  - Also contains meta-data associated with events

- Important to understand the requirements of an event list, and its dynamics, in order to manage events efficiently within a simulation

- In some simulations, event list management may dominate the simulation execution time!

Basic Questions

- Several important questions to consider when contemplating the implementation of an event list:
  - Is the maximum number of events fixed or variable?
  - Is the event list management technique intended for one specific simulation model, or general purpose in nature?
- Two critical operations in event list management:
  - Insertion (also called enqueue) for scheduling an event
  - Deletion (also called dequeue) for removing an event
- Often a tradeoff between these two operations!

Criteria for Event List Management

- Speed:
  - Data structure and algorithms used for insertion and deletion should have minimal execution time
  - Efficient searching is the key (implies sorting, pointers, etc)
- Robustness:
  - Should perform well for a wide range of scenarios
  - Might exploit knowledge of specific simulation model
- Adaptability:
  - Event list management should be “parameter free”
  - Search time depends on length of list and time distribution of events

Example: ttr.c

- Example: time-sharing computer system model called the Think-Type-Receive model (or modified version called Think-Tweet-Read in twit.c)
- Think time: uniformly distributed
- Typing time: uniformly distributed num of chars
- Receiving time: uniformly distribed num of chars

- Notes:
  - Users spend most of the time thinking and/or typing
  - Most of the events in the system are transmitting chars

Example: ttr.c

- Array implementation (unsorted)

Num Users N	Number of Events	Avg Search
5	9,902	5
10	20,678	10
50	101,669	50
100	201,949	100

- Linked list implementation (sorted, search from head)

Num Users N	Number of Events	Avg Search
5	9,902	1.72
10	20,678	2.73
50	101,669	10.45
100	201,949	19.81

- Linked list approach is 65-80% faster!

Many Implementation Choices Possible

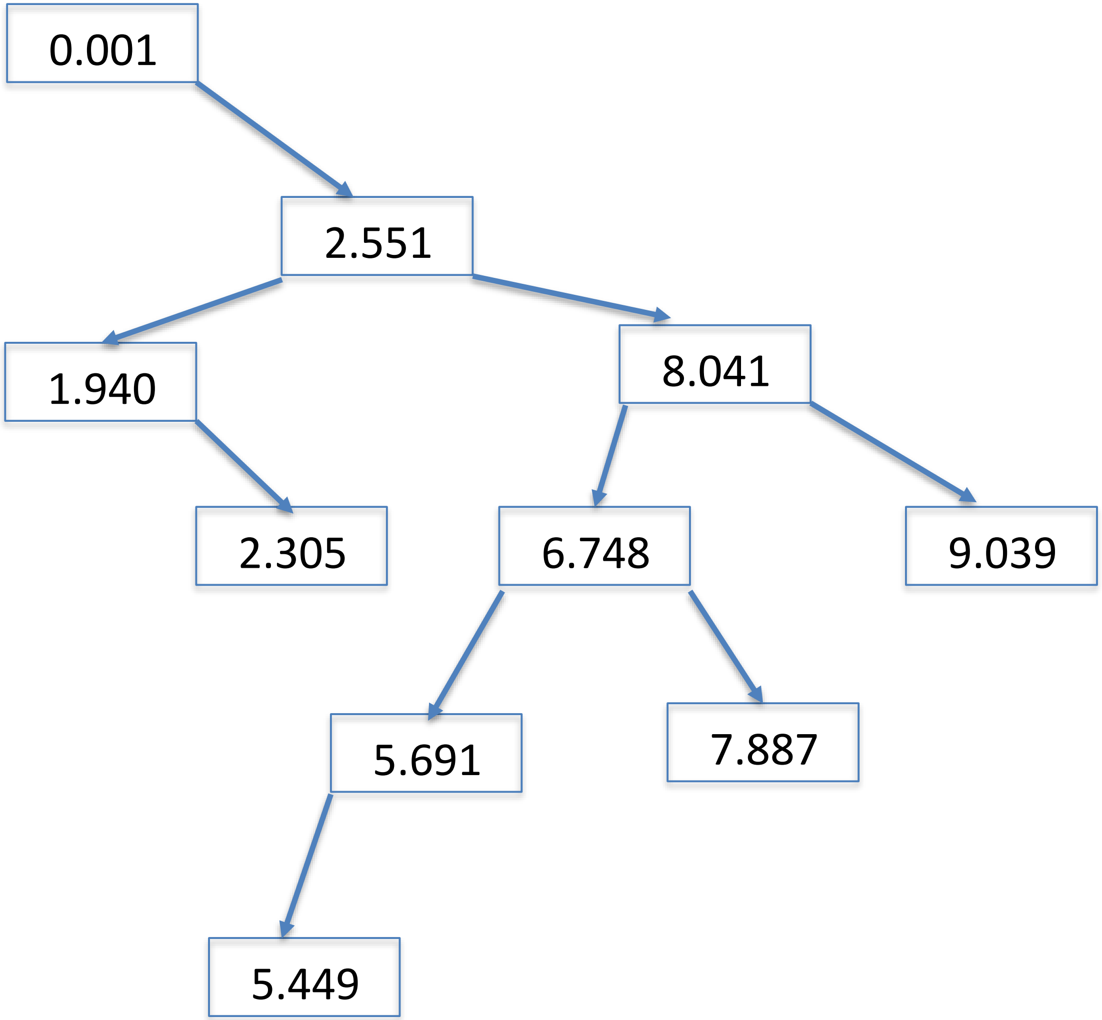
- Array (sorted or unsorted)
  - Suitable for small simulations with < 10 events on list
- Linked list
  - Singly-linked or doubly-linked
- Multiple linked lists
  - Uses k lists, each of which has a subset of the events
  - Could dynamically adjust k to manage average length
- Binary tree
- Heap
- Calendar queue

Binary Tree

- A recursive data structure, where each node has at most two children
- Tree is ordered, with smaller values to the left of the root, and larger values to the right of the root
- Most imminent event would be the leftmost node
- Average case for insertion and deletion is O(log n)
- Worst case is O(n) for n events (linked list)
- Usually enforces a full or a complete binary tree
- More elaborate options include a balanced binary tree or a splay tree

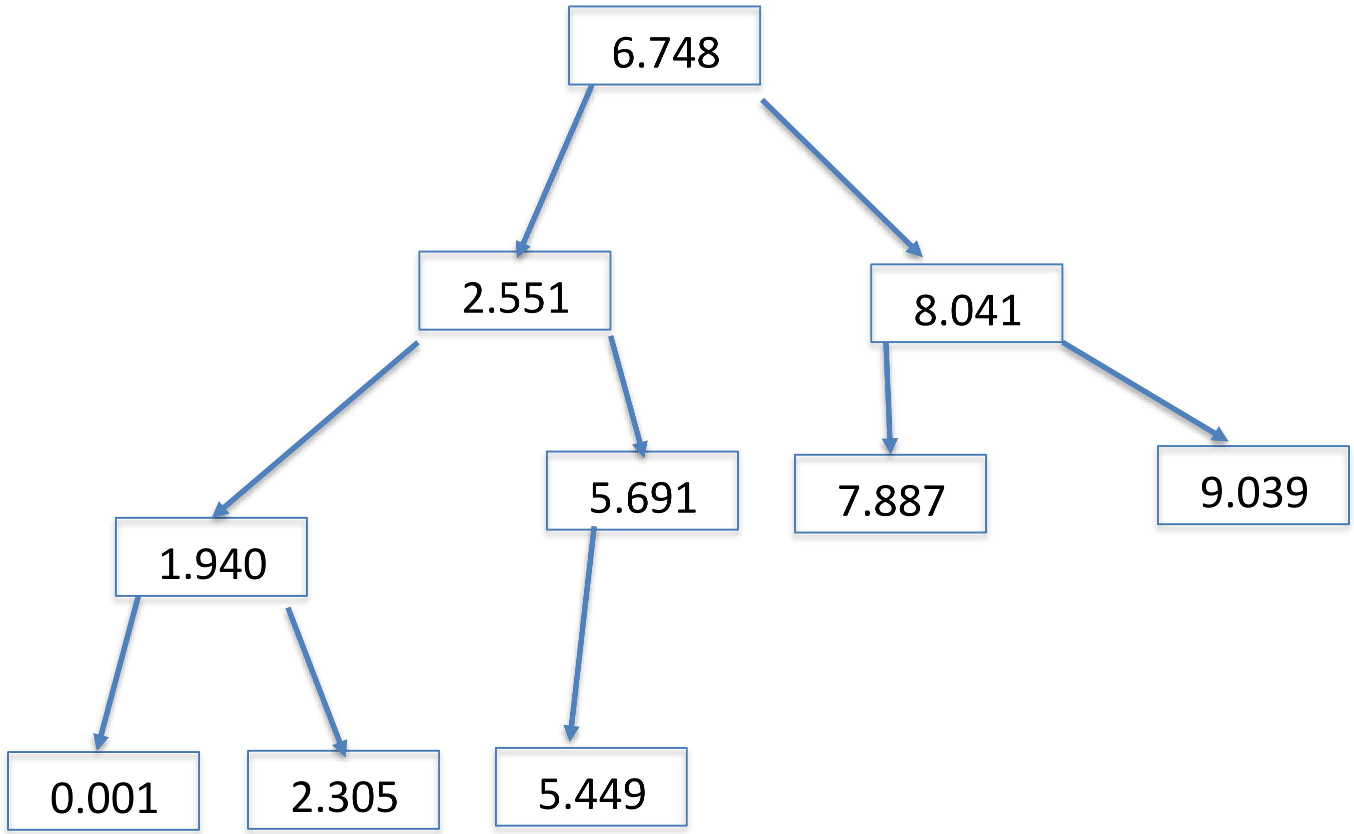
Binary Tree Example

ID	Time
0	0.001
1	2.551
2	8.041
3	6.748
4	9.039
5	5.691
6	5.449
7	1.940
8	7.887
9	2.305



Complete Binary Tree Example

ID	Time
0	0.001
1	2.551
2	8.041
3	6.748
4	9.039
5	5.691
6	5.449
7	1.940
8	7.887
9	2.305

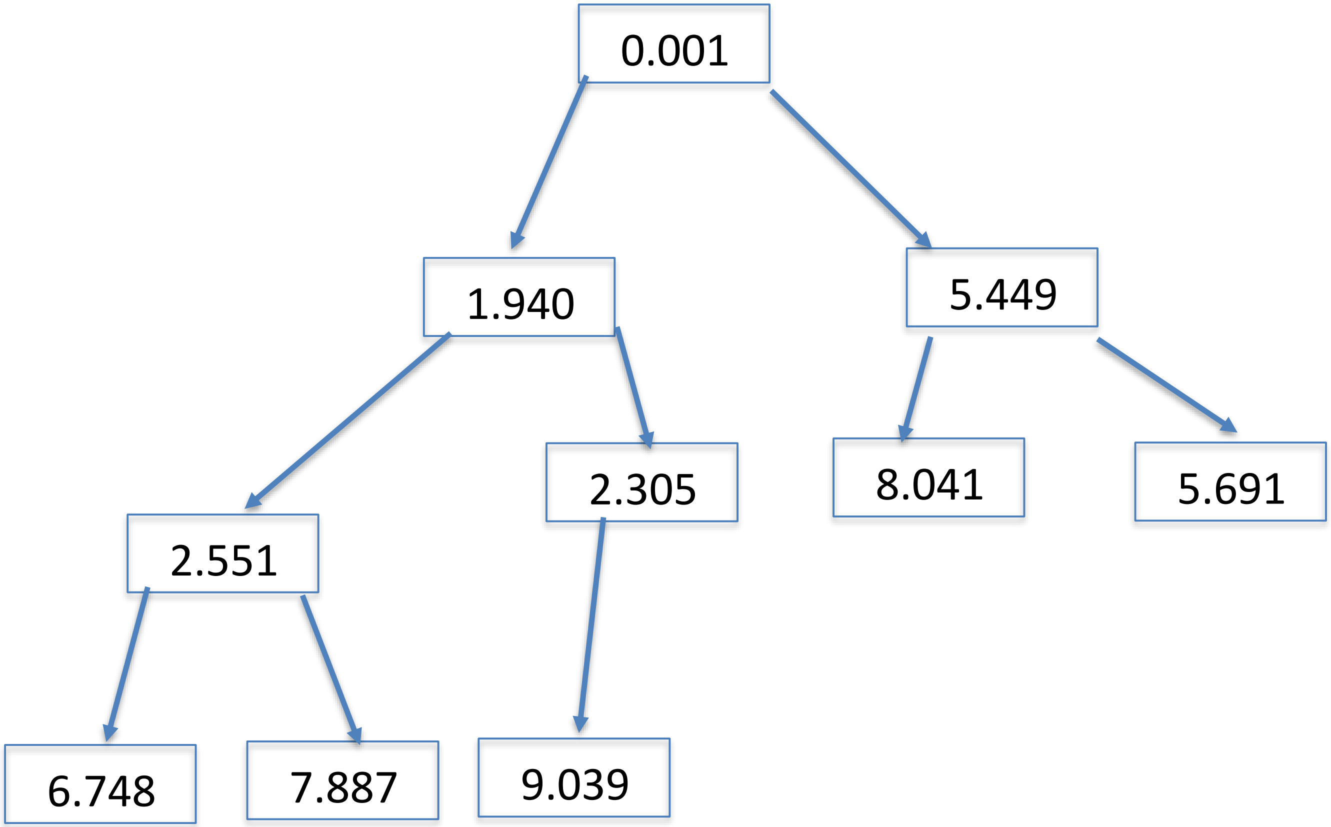


Heap

- A recursive data structure, in which the root node has the lowest event time, and subtrees are heaps
- Most imminent event would be the root node
- Average case for deletion is O(1)
- Average case for insertion is O(log n)
- Usually enforces a complete binary tree model
- Need to maintain heap property upon each deletion and/or insertion
- More elaborate options can balance the subtrees

Heap Example

ID	Time
0	0.001
1	2.551
2	8.041
3	6.748
4	9.039
5	5.691
6	5.449
7	1.940
8	7.887
9	2.305



Hybrid Schemes

- Use multiple data structures, either alternately or in parallel
- Example 1: Linked list and heap
  - Use linked list when there are few events on event list
  - Use heap when there are lots of events on event list
  - Dynamically switch between the two (copying overhead)
- Example 2: Henriksen's algorithm
  - Linked list contains all events on event list
  - Binary tree contains subset of events and times
  - Tree search indexes into doubly-linked list
  - Bounds maximum search distance for insertions (avg case)

UNIVERSITY OF CALGARY

Calendar Queue

- A famous event list data structure [Brown 1988]
- Useful when future events have widely varying times
- Analogy: day planner (day/week/month/year)
- Multiple linked lists, with logarithmic time spacings
- Near future events are on the first (closest) list
- Distant future events are on the last (farthest) list
- Hashing operation determines which list to use
- Avoids cluttering any event list with too many events
- Avg case performance is O(1) for insertion/deletion

Summary

- In complex simulation models, the number of events on the event list might be unknown, and very large
- In such cases, an efficient event list implementation is important for minimizing simulation run time
- Event list coordination is also especially important in parallel/distributed simulation models, because it can become a central bottleneck (i.e., contention)

- We won’t encounter these issues in CPSC 531, but it is good to know when doing (larger) simulations in the real working world!