CS 6301.003: Implementation of Data Structures and Algorithms Spring 2023

Sridhar Alagar
sridhar@utdallas.edu

Administrivia

email: sridhar@utdallas.edu
phone: 972-883-4161

office hours:

Tue 11 AM to 1 PM, Thu 12 PM to 1 PM,
or any other suitable time through appointment
Only on MS Teams.
Link to my office room is on the course home page on elearning

box folder:

all class material will be available in this folder
Link to this folder is on the home page on elearning

TA: TBA

Course Objectives

- Strengthen knowledge of data structures and algorithms
- Extend data structures and algorithms based on requirements
- Evaluate performance of DS&A empirically
- Learn new data structures and algorithms
- Improve problem solving skills
- Improve programming skills
- Improve communication skills

You will enjoy the class if you ...

- Love programming
- Enjoy problem solving
- Appreciate the importance of the proper use of data structures
- Want to strengthen your knowledge of DS&A
- Want to improve your interviewing skills

Knowledge you should already have

Standard data structures, such as lists, stacks, queues, trees, priority queues, search trees, hashing, graphs

Common algorithms for sorting and searching

Design paradigms: divide and conquer, dynamic programming

Fundamental graph algorithms: DFS, BFS, Minimum spanning trees, Shortest paths

Programming skills in Java or C++; OO concepts

Course workload

Short Projects

Long Projects

Test: April 24, 2023 @ Testing center.

Register at testing center ASAP

If not registered 72 hours prior to exam start time, you will not be allowed to take the exam.

No makeup exam for failing to register

Projects

Many programming projects, short and long, will be assigned (Implementation of DS&A, Empirical study of performance of DS&A)

Study project description carefully, especially input/output specifications. Each project may have a starting code base and acceptable reference resources.

Do not deviate from I/O requirements. If starter code is given, do not modify the signatures of public methods.

Sample inputs and outputs will be given for long projects. But you should create test inputs on your own, trying to cover all cases.

Short Projects

Assigned weekly. Work in pairs to solve one problem each week.

In addition, implement additional practice problems given.

Most of the short projects do not carry excellence credits.

No late submissions will be graded (without prior approval).

You can resubmit before the deadline. Only the final submission will be graded.

Long projects

Bigger projects with longer deadlines (usually 2-4 weeks). Size of team: 4.

Algorithms needed for the projects are taught in class.

Submissions can be revised before deadlines.

Extra or challenging work will be given for extra credits.

Each work can earn 0-1 excellent credit (EC), based on code quality, correctness, and running times. Submissions failing many test cases do not get any EC.

Groups

Short projects **must be** done in pairs.

Long projects **must be** done in teams of 4 students each.

Please form your own group for LP.

Group members for SP will be randomly assigned

Any requests to change groups must be approved by the instructor. Seek help from instructor if you are unable to resolve problems within your team.

On code quality

Write production quality code

Follow software engineering principles.

Main goals:

1. Safe from bugs
2. Easy to understand
3. Ready to change

Checkout MIT course on Software Construction (web.mit.edu/6.031)

On code quality

1. Give meaningful names to all identifiers (names of classes, methods, variables). Organize code in each class: fields, constructors, interface methods, nested classes, helper methods, miscellaneous.

2. Format code to be readable. Limit no. of lines in a method to 1-2 screens.

3. Document your code with comments: e.g., input, output of functions, loop invariants, assertions, global variables, preconditions, exceptions. Follow Javadoc standard: https://en.wikipedia.org/wiki/Javadoc

You will lose points if you do not follow Javadoc standard

On code quality (continued)

4. Avoid dead code, debugging code, unneeded or unused variables, unnecessary output, “// TO DO” comments left unfilled.
5. Learn proper use of static and non-static methods.
6. Learn usage of public, protected, private.
7. Avoid code repetition (possibly due to cut-and-paste). Code duplication may indicate that a new function needs to be created to perform this common task.
8. Minimize usage of global variables. Use global variables mainly for constants, that all methods use. Avoid passing return values of functions using global variables (i.e., avoid “side effects”). Follow functional programming.
9. Create test inputs (unit testing) for all methods and classes.
10. Conform to input/output specifications.
11. Use standard solutions whenever possible (using data structures or encryption functions from libraries), rather than implementing them on your own.
12. Use simplest data structures with the desired running times.

On code quality (continued)

13. Avoid “clever” encoding schemes: “12#25#498” to encode three integers {12, 25, 498} as return value of a function.
14. Avoid strings and hashing, except where really needed.
15. Learn proper use of generics; avoid @SuppressWarnings.
16. Develop code that is type-safe, that will not generate type errors at run-time.
17. Do not catch exceptions without writing code to handle it. It is better to declare your function as “throws SomeException” than to write “catch (SomeException e) { e.printStackTrace(); }”.
18. Include names of all members of team, date of creation and changes, purpose, and other release notes in all source files. Cite sources of code that you used from outside sources.

Grading of projects

Projects will be evaluated by (manual) code review and by executing them on (large) test cases.

Students are expected to follow good software engineering practices and write high-quality code.

Each project should also be accompanied by a readme file (txt) and a brief report (txt, pdf, doc) that summarizes results, cites sources.

Projects may be penalized for poorly written code.

At the discretion of the instructor or TA, you may be asked to come and show a demo of the project.

Making an “A”

Earn at least 3 excellence credits.

Score an average of 90% or more in projects.

Score at least 80% in test

Attendance, participation, recording

Regular attendance is strongly recommended
- Direct correlation between attendance and grade

Participate in classroom discussions
- Makes the class lively
- Enhances understanding
- Don’t be shy. No question is stupid

I will record the lecture on Teams and make it available for viewing later
- General -> Files-> Recordings -> View Only
- No guarantee
- Will stop recording if attendance is low

Discussion Board

We will use the discussion board in blackboard to discuss project related matters

Do not send me an email regarding this

Post your questions in the relevant thread.

If no relevant threads, create a new thread

Feel free to reply to others’ questions

Don’t be shy. No question is stupid

Subscribe to the forum to be notified when there is a new post

Project submissions

All projects must be submitted on elearning. Do not submit large test data files.

Deadlines for projects will usually be at 11:59 PM on Fridays/Sundays.

Each project should be submitted as a single zip file. Do not use other formats like 7z. Include a readme file (txt). Projects submitted as individual files or in other formats will not be graded.

Your files should unzip to a folder whose name is your netid (e.g., exa123456). Do not create additional subfolders. Do not place any files outside this folder.

All submissions can be revised (more than once, if needed) before their deadlines. Only the final submission before the deadline will be graded.

Use your netid as the package name for all your projects.

Do's and don'ts

You may interact and learn from any of these sources:
- Instructor and class notes
- Students in our class and the class forum on elearning
- Any textbook on data structures and algorithms
- Lecture notes made available by instructors (world-wide)
- Wikipedia and other web sources

Don't ask for help in writing or debugging your code.

Don't share your code until after the semester is over. Set up your github account to use a private repository. Otherwise, your code is accessible to others.

Don't use code from internet (or other) sources that is not explicitly approved in the project description.

Honor code

All students shall maintain the highest level of academic integrity and honor.

All sources and collaborations must be acknowledged in your project reports.

Code found to be plagiarized (from other students or from web sources) will be referred to OCSC for disciplinary action.

For more information, see the URL:
https://www.utdallas.edu/conduct/integrity/

Ethics

Cheating in classes makes no sense. Would you play a game in which you have 0.01% chance of winning $1, and if you lose, you pay $1M?

Grades that you get, will have small consequences now, and no consequences in the long term. Knowledge you obtain & habits you develop, will serve you for life.

Things that are worthy in life: knowledge, good quality work, accomplishments, reputation, family, friends.

A good reputation takes a lifetime to build, and one act of dishonesty to ruin it.

Don't cheat. Don't help friends cheat.

Binary search

Input: Sorted Array of integers A[0,…,n-1] and target t
Output: result = -1 if t is not in A else A[result] == t

Binary search

Input: Sorted Array of integers A[0,…,n-1] and target t
Output: result = -1 if t is not in A else A[result] == t

Binary search

```
/**
* Checks if target is in A using binary search
* @param A A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* @param target search for target in A
* @return -1 if target not in A else res st A[res] == target
*/
public static int binarySearch(int[] A, int target){
       lo = 0;
       hi = A.length -1;
       while (lo <= hi){
              mid = (lo + hi)/2;
              if (A[mid] < target)
                     lo = mid + 1
              else if (target < A[mid])
                      hi = mid – 1
              else
                     return mid; // found
       }
       return -1;
}
```

Binary search

```
/**
* Checks if target is in A using binary search
* @param A A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* @param target search for target in A
* @return -1 if target not in A else res st A[res] == target
*/
public static int binarySearch(int[] A, int target){
       lo = 0;
       hi = A.length -1;
       while (lo <= hi){
              mid = lo + (hi - lo)/2;
              if (A[mid] < target)
                     lo = mid + 1
              else if (target < A[mid])
                      hi = mid - 1
              else
                     return mid; // found
       }
       return -1;
}
```

Binary search variant

Input: sorted Array of integers A[0,…,n-1] and target t

Output: return index i such that A[i] <= t < A[i+1]

if t < arr[0], return -1; if t >= arr[n-1], return n-1

Loop Invariant

```
// precondition P
init // initialization code
// Loop invariant I
while (C) {
        S        // body of the loop
}
// postcondition Q
```

Precondition P is a proposition that is true before executing the loop

In binarySearch() method the precondition before the while loop is:

the array is sorted – A[0]<=A[1]<=…<=A[n-1]

Loop Invariant

```
// precondition P
init // initialization code
// Loop invariant I
while (C) {
        S       // body of the loop
}
// postcondition Q
```

Precondition P is a proposition that is true before executing the loop

Postcondition Q is a result after executing the loop given P

In binarySearch() method, the postcondition after the while loop is:

true if x is in array; otherwise, false

Loop Invariant

```
// precondition P
init // initialization code
// Loop invariant I
while (C) {
        S        // body of the loop
}
// postcondition Q
```

Loop invariant is a property of the program which is true (given P):

1. before the execution of the loop (after init)
2. after each iteration (execution of body S once)
3. after the loop terminates (condition C is false)

In binarySearch() method, L.I is

x is not in A[0…lo-1] and x is not in A[hi+1…n-1]

For more on invariants read:

1. https://www.cs.cornell.edu/courses/JavaAndDS/loops/01aloop1.html
2. Programming Pearls, Second Edition by Jon Bentley. Available for free through safari online

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*             if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
       //init
       //LI:
       while (){
       }
       return ;
}
```

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*              if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
      //init
      l = 0;
      h = A.length -1;

      //LI:
      while (){
      }
      return ;
}
```

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*             if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
      //init
      l = 0;
      h = A.length -1;

      //LI: A[l-1] <= t < A[h+1]
      while (){
      }
      return ;
}
```

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*             if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
      //init
      l = 0;
      h = A.length -1;

      //LI: A[l-1] <= t < A[h+1]
      while (){
            m = l + (h-l)/2
            if (t < A[m])
                  h = m – 1
            else
                  l = m + 1
      }
      return ;
}
```

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*             if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
       //init
       l = 0;
       h = A.length -1;

       //LI: A[l-1] <= t < A[h+1]
       while (l <= h){
              m = l + (h-l)/2
              if (t < A[m])
                     h = m - 1
              else
                     l = m + 1
       }
       return ?
}
```

Binary search variant

```
/**
* pre-cond: A[0]<=A[1]<=…<=A[n-1] where n is the length of A
* post-cond: index i such that A[i] <= t < A[i+1]
*                if t < arr[0], i = -1;  if t >= arr[n-1], i = n-1
*/
public static int bsearchv(int[] A, int t){
       //init
       l = 0;
       h = A.length -1;

       //LI: A[l-1] <= t < A[h+1]
       while (l <= h){
              m = l + (h-l)/2
              if (t < A[m])
                     h = m – 1
              else
                     l = m + 1;
       }
       return h;
}
```