**Summary:** This is the worklife cheatsheet used to guide worklife structure. This is the central document.

**Todo:**

~~1. Todo: Make picture. \te{20 minutes}.~~

~~2. Todo: Make goals sheet \te{1 hour} \pr{}.~~

3. Todo: Integrate goals sheet with worklife todo \te{1 hour}.

4. Todo: Connect picture with implementation = Todo: connect picture with documents + ~~Todo: be sure to connect week plan and experiences sheet in document~~ + Todo: connect picture with mindmap \te{40 minutes}.

5. Todo: Make steps to construct picture, Add steps to clear, reset button \te{20 minutes}.

6. Todo: Add summary \te{10 minutes}.

7. Todo: Integrate business document summary into section 3 details \te{15 minutes}.

8. Todo: Special notes on code \te{10 minutes}.

9. Todo: integrate real experiences into experiences sheet

10. Todo: figure out how to integrate social with otherwise.

11. Todo: integrate with courseware, e.g., mechanical, electrical, computer

12. Todo: Experiences and then integrate - starting from yesterday’s tm and movie:

13. Todo: Consider adding self portrait below at end of second page:

14. Todo: Put in management infrastructure

15. Todo: Add in TM notes:

16. Todo: Top-level picture view:

17. Todo: Goal: Working version / production:

18. Todo: Goal: Integrate experiences into sheet

19. Todo: add ring:

20. Todo: insert links in word documents at text ‘Document extension’

21. Todo: add in documents.

22.

Final product

Building

**british:**

*Habits: intro*

*Formation and Themes:*

Be natural, and then work hard too.

PC/P: learn yoga to its finest.

S and m: / Think about Hilton Garden Inn, where you felt it first. MIT.

Reset state: clear mind

Start state: ohm world is s and m

Pin down center:

Don’t lose yourself, but also improve yourself.

How this was formed: Keep it simple, research questions, integrate, comm, logic.

History+ levels: Maturity: Amanda: | Money: | Stress relief: | Gropp: | Mgmt: habits-summary | PNA: |PCA: | OOP: |Dad: |Man from internet |Experience: mom|

Cross-link:

Habits summary is a preview of maturity, just before 3d cubes. Helps to smooth cubes out. Should always be thinking about the worklife from starting man from internet and dad, and with management on top. Static can be smoothed out through maturity lines.

MPI book: stick.

MPI Implementation: bschool.

MPI Standard: engineering, social intelligence.

*Rules:*

Habits need to pervade everywhere throughout.

Vertical in wl. Horizontal in wl.

Rule propagation: example: confidence – think about how someone else responds.

Use logic to make things follow. Example: keys, wallet, cell -> backpack, etc.

Experience Generalization to rule. Example: clothes in India, generalize to many things.

Experience transfers to other experiences.

Recurse: this generally happens naturally, but just know that it can be done.

*Prioritization:*

Work: Code: 40, Data:30, Admin: 10, Writing/pres: 20

Mgmt: Goals: 30, month: 10, weekPlan: 30 , daily todo: 15, 10: prioritization, 10: scheduling

Comm: L0: 40, L1:25, L2: 15, L3: 10, L4+L5: 10 Wl cheat sheet.

*Implementation:*

Practice: remember calculus exam study. Once you practice and memorize, it becomes in-grained in your head in working memory.

experience-sheet of paper with situations, mgmt: ppt and latex. use 4 sheets as a guide esp. when static comes in. Look at implementation picture. Habits(expand on this – needs to pervade all). When applying or doing an activity, use number system to identify what part it belongs to. Independence/Interdependence.

Discuss how every week should be handled in terms of week plan doc and goals doc, along with sheets.

Independence:

1. proactive: circle of influence vs circle of concern --
2. begin with the end in mind : goals.
3. prioritize: urgent/important, deadlines.

Interdependence:

1. Think win-win: --
2. Understand, then be understood: --
3. Synergize: --

Quality:

7. Sharpen the Saw: practice, re-evaluate system, read a book and come with it at different angle.

Concrete implementation: (in terms of files on comp)

Sharpen the saw.

*General wisdom from going through the sheets:*

Money and work are close, but work and happiness are even closer. God and happiness and Social intelligence and maturity are connected.

Money stays constant, and is related to static. Comes through experiences. Sayings.

**Follow calendar of events (full implementation):**

General Todo:

1. Prioritize the above.
2. Break each into smaller pieces.
3. (Maybe) Look at competition, and do better.
4. Make sure to maintain minimally working version of you.
5. Have full pathway of entire sheet intact (can be multiple different lines).

3. Do these in sittings when there is nothing else to disturb.

6. One thing at a time, and add.

7. Order the additions of different things to add.

*(note: add more, org better, make plan to read newspaper – gen Knowledge + 52 weekends nytimes books).*

TODO: (1. Shorten this 2. add more to comm 3. Make Work section better 4.)

Quick Todo: 1. Fix the experiences section, going through experiences. Add real reliability points. 2. Add real happiness – 3 hours. 2. Figure out 3-D; 3. Prioritize worklife cheatsheet. 4. Add text to guide each section. 8. Add tm sheet, w/ social int sheet – Tuesday. 5. Add text to guide each section in mgmt. 6. Add worry management to happiness in habits. 7. Consider margin changes for more space in wl-cheatsheet9. ../thePast/maturityVersions: Clean-up text, make accurate, add intermediate levels; 10. Read habits – confidence integration; 11. Order content, make any points clear, order/structure content, connect all points together. 12. Add in happiness wiring, and connect all the way down to real-time confidence 🡪 static to dynamic. 10. Make sections in above more clean. 11. Fix the section for gen knowledge. 12. Integrate summary with cheat sheet better.

*Mgmt (TODO: 1. check ordering, including where leadership goes- 2. Put below personal development? 3. Figure out where leadership goes 4. Connect to either Pers. Dev. Or Management endpoint to comm.) 🡨 put in margin next to space*

*God:*

Note past situations from childhood that could create awkward situations and define relationships. – could put this in experiences.

Remember that all acts as guide, especially at the top and bottom (gut /randomness is ok). Randomness can be used to your advantage.

**Personal maintenance:**

* 1. *Monthly:* change lightbulbs, dentist appointments, get meds, replenish wallet, haircut. *Weekly routines:* garbage, dishes, vacuum, decide clothes for week, iron clothes, situations, fill gas + grocery, clean out backpack, check nails, clean laptop screen + outside, clean car, get clothes for week.
  2. *Regular Routines (habits)*: bed, teeth, empdw, stretch, food, clean kitchen floor, exercise, email, dailyTodo, shower (hair, shave (sides, under, chin, moustache, vuln: mouthsides, mouthtop, belowlip), eyes, ears, neck, armpit, feet), towel, meds, belt on+pants high, comb/hair spray, lotion, cnclr, Work, Night: mail, clndw, charge phone/laptop, workout, check for events + calendar, emails, Add link.
  3. *Static:* link to 4 detail sheets:
  4. Habits summary *🡪 Ecological Intelligence -* impact region, economical.
  5. *Happiness (apply to happy thought): urbana house. If you think about girl, think that there’s no doubt ...*
  6. *UnderstandWorry:* emotional intelligence: whatAreEmotionsFor+ScienceOfEmotions+Emotions / Intellect disparate, self-awareness/real-time awareness, anger+anxiety, delayed gratification + positive thinking + Hope + Flow-confidence / learning and Flow.
  7. *StopWorry (apply to each worry):* day-tight, what’s the worst, knowthefacts, prob/cause/all sols/best-sol, don’t saw sawdust, cooperate with inevitable, don’t imitate others, don’t get even, meditate/exercise, (get books on self-confidence). If someone asks question, don’t think negatively.
  8. *Work:*
     1. Sincerity, ethics and honesty, Diligence, work 80 hours a week, enjoy what you do.
     2. Prior work, baseline data, define problems, contributions, theoretical analysis.
     3. (P)NA. Code/Impl, Apps, Research Questions + Results, Profiling to Explain Why (App, hardware/PAPI, metrics). (P)CA.
     4. Project management & collaboration (git for code, cmake, README).
     5. Writing skills (latex for presentation, svn for paper collab, elements of style, persuade through intro, no formatting mistakes, envisioning information, presentations, posters).
     6. Awareness of technology to be using in your field / Awareness of developments in your field – related work.
     7. Time management at work (meetings, schedules, deadlines), Administrative.
     8. Marketing yourself and your work – title, abstract (dry/factual), intro, put info on website/linkedin;
     9. Work:rel: admin:
     10. Companies: BNL (D.E. Shaw, GS), PNNL (Yahoo, Google, Palantir, fb, cray), Stanford, LLNL. Keep in mind that all companies are good (it’s what you make of it).
     11. Context/Weather/environment:
  9. *Personal Development:*
     1. Depth of Gen. Knowledge / hobbies: Tennis, Piano, Bridge, chess.
     2. Leisure: automotive(rolls/jag/rangerov/Lincoln/ford, Cadillac/GMC, porsche/Bentley/merc/bmw/chrys/jeep/fiat, Ferrari/masa/alpha, lexus/Toyota/acura), tv shows (Office, MF, Friends), songs.
     3. Financial: economics, stocks, gas prices.
     4. Current Events: Books (World is Flat, Hot Flat and Crowded), magazine (==), News (work: wired, gen: nytimes),
     5. Logistics: city structure, airport structure, general driving knowledge, grocery store structure, gps, subway systems.
     6. Geographical: U.S. map, europe map, city maps, airports, freeways, quirks about places, subways, landmarks, driving.
     7. Work/professions: lawyer, engineer, scientist, educator, doctor, hr.
     8. Geo-cultural/political: cultural, political (liberal vs. conservative, practical vs. theoretical, gen. vs. spec.), climate/weather.
  10. *Management:*
      1. Health: Weight (chew food, avoid eating large portions when avail., Exercise: lift, jog, yoga), Avoid sickness, Acne, Dent (floss, whitening).
      2. Money: southwest fares, reimbursements, rent, uiuc bill, cash, paycheck, account for cash, owing ppl money, points for air travel.
      3. Time: Look at calendar for each week, write action items in iPhone todo, have monthly plans, find social events/tech events, prioritize.
      4. Spaces: Jean pockets (KeyWalletCell), Backpack (don’t put on floor or bed), office, laptop (clean outside, org folders), room (bed, closet, floor), car (outside, carpet, inside glass, seats, panels), kitchen, bath (floor, towel bar), lock doors before leaving.
      5. Appearance: belt on, pants high, XYZ, deodorant, gum, laundry stains, shirt not inside out, iron shirts/pants, button shirt, matching/non-torn socks, plan weekly wardrobe, check for clean teeth and face, non-inverted shirt collar, shirt not half-tucked, haircut/nailcut, hairgel, dinner etiquette.
      6. Facebook/linkedin/twitter/instagram/Whatsapp: know software updates, profile up-to-date, check frequently. Cover photo: specific to your interests, blends with profile, change seasonal/events – keep stable after some time, birthdays.
      7. Leadership: Listening, critical thinking, giving feedback, time management, planning and implementation, organization and delegation, facilitation, motivation, mentoring, team building.

**Social interaction:**

* 1. *General principles:*
     1. Communication organization:
        1. email: check recipients, no double-sends, check spacing/formatting, spell check, check for internet cxn b4 sending, don’t send angry emails.
        2. gchat: type outside chat box, send quickly and have a roadmap for conversation, be clear, check spelling.
        3. text: short texts, quick replies, no getting overworked about someone’s confusion about you.
        4. phone: articulation, speak loudly, leave short vmails.
        5. skype/video: find a place with good lighting, get audio/video tested, make sure background noise is out.
     2. General rules of interaction:
        1. Level 0+1: Think positively about interaction: No getting overworked abt other’s remarks, stop beating yourself up, be humorous / Be Honest, ethical to yourself and others | Act cozily | Reliability / Consistency / Confidence /Shout.
        2. Level 2: Be logical, make sure you make sense / avoid random thoughts that come to mind / structure thoughts and check for non-sequitors.
        3. Level 3: Be intentional / Think before you speak / integrate with top-level point.
        4. Level 4: send message in way that others will easily receive it: *howtosayit*: grammar/picture errors, emphasize at the end, that vs. which, check for gen. rules (don’t use abbreviations), *bodylanguage*, *vocalvariety*.
        5. Level 5: *persuasion, avoid I: a. give honest and sincere appreciation b. don’t criticize, condemn, complain c. arouse in the person and eager want:* Be happy with others: Avoid mis-interpreting others’ intentions / don’t get mad when someone doesn’t understand you | avoid neg. thoughts | Identify common things | Listen to and Read people (body language, facial expression) || *empathy, wired to connect:* Situational awareness / Focus on one thing, and do that thing / Make eye contact / don’t be interested in random parties or randomness outside / emotional intelligence | Give space, avoid being over-intrusive.
  2. *Specific settings:*
     1. Technical presentations: find ways to break ice and avoid nervousness/think of audience as rooting for you, organize speech (be linear), get to the point, how to say it, your body speaks, vocal variety, research your topic (know content), use visual aids/materials (concrete?), persuade, inspire/connect, entertain/joke.
     2. Social gatherings/conventions (table topics): Be resilient to others words, think of words before speaking and making your point, punctuality, prep clothing, know schedules and email others in advance, know interactions that may come up, know what to go to/where to be, identify common ground in group and situational awareness, connect with the overall situation (mtg theme).
     3. Meetings/hangout/interview (conversation): prep for what to say, avoid going into things that generate unneeded work.
     4. Relationship setting: Be on lookout / think positively / icebreaker, Planning dates (make reservations, tell ahead of time what’s going on, don’t make things too complicated), Have something interesting + unique to say, show interest in specific things, Connecting (Find common ground, show compassion, show interest in the other person’s activities), Express emotions when you have them (when you like someone, tell them), connect about emotions.
  3. *Social Intelligence:*  --- Matrimony: 🡨 habits summary
  4. Mannerisms to have and to avoid: project voice, talk slowly, lips relaxed, smile, no frowning, eyebrows up, relaxed hands, avoid over-smiling, avoid moving arms, hands in pocket or to the side, avoid crossing arms, back straight, walk with brisk pace, don’t lean on places, walk straight line without feet scrubbing floor, give hugs tightly, shake hands thoroughly, no looking at phone, avoid bodily noises (no teeth clicks, no nose sneezes, no farts), don’t bump into people, clean trash behind you, fashion.
  5. *Working with Emot. Int (fix):*experience: emotional, social, general if/then, reliability, maturity/practicepoints/positivepoints, rule-experience, specific if/then, security, fault-correction, fault-tolerance.

**Application of Pers. Maint. And Soc. Interactions to Real Situations:**

Method of learning, and validating model

1. Identify situation, analyze what you did right and wrong
2. Write it down in docNotes
3. Practice it again in situations
4. Figure out where it fits in workLife -- connect with big picture
5. Unify, simplify and generalize
6. Some things from situation carry over to other situations
7. Organize according to general rules of interaction
8. Identify upcoming events and apply based on general rules + specifics
9. Prioritize the items

***Any situation:***

* belt, pants high,
* check wallets/key/cell , shave
* iron shirts
* deodorant
* no stains
* go to bathroom
* check concealer
* comb hair
* back straight
* think of points / objectives
* driving/transport

***Rel: Weekend Date:***

* go running
* Emo. Int.: identify points of nervousness
* Self-awareness:
* Self-regulation:
* Motivation:
* Empathy:
* Social Skills: influence, communication, conflict management, leadership, change catalyst, building bonds, collab, and cooperation, and team capabilities.
* If she’s poking fun at you for your weirdness, don’t get stressed about something someone said, or something you said, don’t get nervous, don’t get worked up on an opinion.
* Comm: have topics to talk about to open conv. (plan this out), eye contact, avoid distracting mannerisms.
* Comm: inspire: find common ground on little opinions, don’t talk about weather or generic things like work/traffic/news, find a place to spend time together alone.
* Mgmt: Time: Plan events for yourself a week in advance.
* Mgmt: SharedTime: Tell plans to other person, make sure it works for them.
* Mgmt: Time: Talk to parents about it / talk to friends about it 3 weeks before hand.
* Mgmt:appearanceVuln: no buttcrack, don’t smell, teeth/lips clean, socks clean/non-torn.
* Comm: lips relaxed, smile/avoid looking stressed, back straight.
* Mgmt: Appearance: Figure out clothes to wear each day, concealer
* Mgmt: Spaces: Make sure phone fully charged for each event
* Comm org: make sure to send to correct recipients, be clear in texts during coord.
* Mgmt: Logistics: know directions, know how to get to/from places.
* Mgmt: appearance: Dinner etiquette at meals.
* Error-detection: watch for
* Error-prevention:
* Error-correction:
* fault-tolerance:

***Rel: Skype Date:***

* Comm org: Make sure media is fine, find good lighting, have an intro.
* Comm:org.+intentional: have goals and outcomes, Figure out topics to talk about which connect.
* Comm: Make clean exit, know how to finish convo, keep convo to 30 mins.
* Comm: Mgmt: Ensure no one is around.
* Comm: Mgmt: Get the timing clear (time zones, etc.)
* Weddings, dinner parties, bar meetups, house parties
* Don’t laugh too much at something, to give the wrong impression

***Meeting with a friend at a bar:***

* Comm: Have specific items to talk about and catch up on
* Comm: Know answers to direct questions.
* Comm: Warn yourself to not be too intrusive.

***Meeting person X at work:***

* make sure you don’t smell.
* avoid farts, go to bathroom beforehand.

***Meeting advisor***

* have clear points to communicate
* make preview before hand
* be linear

***Meeting with several friends:***

* Comm: don’t be too intrusive.
* Have opening:
* Comm: don’t be excessive.
* Check shirt and make sure things are in order with quick changes
* Avoid negative points.
* Don’t say something and modify.
* Know what to do when you meet certain people.
* Don’t be too intense, esp. if someone is talking about something familiar.
* Don’t be the center of attention.
* Quick points without
* Say a few things, don’t go silent.
* Don’t get worked up when someone says something
* Don’t try to get attention in a group
* Eat with proper etiquette
* Pay attention to what’s going in a group setting, if questions, then ask
* Have closing

***Group Meeting:***

* Mgmt: send email beforehand to discuss what to do
* Make sure to discuss results, action items
* Comm: don’t discuss too much

***Full-body /all-hands Group Meeting:***

***Party at a bar:***

* Know directions, figure out timing for how long it will take
* Find people to go with
* Get the right clothes to wear, make sure clothes are ironed
* get cash

***Party at friends place:***

* Mgmt: find directions to house, check that you can get in, if shared community
* Comm: Be quiet and don’t impose
* Do what others are doing.
* Offer to help
* Don’t overstay your welcome

***Work social hours:***

* take badge with you
* avoid over-eating
* talk to people about your work
* clean shirt
* go with clean jeans
* avoid looking like you need the food

***Wedding of a close friend:***

* Mgmt: know clothing
* Have a list of sub-situations
* Know the agenda without intruding on others
* Don’t try to get attention from others

***BMM Convention:***

* Check events beforehand
* Don’t try to get attention from others
* Clothing

***SC conference:***

* check events beforehand
* clothing

***Lab Presentation:***

* don’t make assumptions on what people know
* be ready to answer questions easily

***Coding:***

* Follow code complete coding principles
* Think about big picture
* Compilation procedures
* Google when errors occur, and when confused
* Take breaks

***Results Collection/Experimentation:***

* envisioning information.
* talk about which research questions you are answering.
* think about research question.

***Paper writing:***

* Correctness in techniques, problem and results, Make sure about words making sense
* Structure is logical and flows without forward references
* CommInWork+Mgmt: commit changes and let people know
* Overall structure placement is good, submission-ready.
* Consistency across sections
* Use the right vocabulary
* CommInWork: Writing style

Specific Scenarios:

***Coming back home from the car:***

* Put keys, wallet, cell phone in the right place

***Email exchange for work:***- Check email formatting

* Use good style – read how to write good emails

***Someone asks “how old are you?” or personal question. In a group setting where friends getting attention and you’re not.***

***If someone gets angry at you:***

***Meeting a girl who likes you:***

***What do you work on?***

***Anytime:***

***Applying experience to above:***

***Implement the experience-learning:***

***Fluff( ) ; implement experience-learning.***

***Fluff2***

***Reliability : /error correction spinning with experience – spinning with practice.***

***What one person might say can apply to another person, though to a different extreme.***

***Make sure situations are clearer (why didn’t I bring the lexus?)***

***Cheating: teacher thinks cheating a regular failing student, but student doesn’t believe.***

Work: Working with Tami/secy,

Social: Honda dealer car.

Work: Gropp, bronis, …, garzaran, padua, torsten, todd,

Family: Dad, mom, Atul, Ridhima, Rakesh Uncle, Pinki Auntie,

Friends: Rishi, Chris -

Collaborators: Amanda, .. Costin, …,

Related Work: DPLASMA, rice-sarkar, locality,

Competition: … , …,

Work status: (layers): Livermore scholars, students

Social status: Indian, Brahmin, Marathi, …

Bosses: advisor, students,

Relationships: sateja, avani, others in queue – remember asm, experience.

Fine-grained.

Real-experiences – Paris.

God.

HIVE, count your blessings, harness the positive thoughts, alleviate key points of stress by talking them through, YOLO, dwell on happy thoughts (get books on depression). Urbana house, sunysb, -, highschool grad; asm house.

1. Worklife-cheatsheet - ppt slides
2. WeekPlanDoc ppt + weekplanDoc latex.
3. Mini-notebook.
4. Excel spreadsheet of jobs applied to
5. Experiences Sheet.

Habits: Circle of inf and circle of Control

**Happiness:**

(read intro)

entertainment speech (jokemaster rules).

Emotional Intelligence:

Facts and Logical structure:

Real examples:

Connect with other parts:

Simplify (compress):

Prioritize examples, and integrate with wl-cheatsheet:

Integrate with Toastmaster’s:

Look through Thesis:

**Stress Reduction / Anxiety / Meditation:**

1. Live in day-tight compartments

2. If Trouble backs you in a corner:

a. ask yourself, what is the worst that could happen?

b. Prepare yourself to mentally accept the worst if necessary

c. Calmly try to improve upon the worst, which you have mentally tried to accept.

3. “Those who do not know how to fight worry die young”

4. Get the facts. “half the worry in the world is caused by people trying to make decisions before they have sufficient knowledge on which to base a decision”

5. After carefully weighing all the facts, come to a decision

6. Once your decision is carefully reached, act! Get busy carrying out your decision, and avoid anxiety of the outcome

7. Business worries:

1. what is the problem?
2. what is the cause of the problem?
3. what are all possible solutions?
4. what is the best solution?

8. Crowd worry out of your mind by keeping busy.

9. Don’t let little things – small bugs/beetles - ruin your happiness.

10. Use the law of averages to outlaw your worries. “What are the odds against this thing happening at all?”

11. Cooperate with the inevitable. If you know a circumstance is byond your power to change or revise: “It is so; it cannot be otherwise”

12. “stop-loss” order on your worries .

13. don’t saw sawdust. let past bury its dead.

14. Fill our minds with thoughts of peace, courage, health and hope. We control each thought. Our life is our what our thoughts are.

16. Never get even with our enemies.

17. Don’t expect others to be thankful to you.

18. Count your belssings not your troubles.

19. Don’t imitate others . “envy is ignorance”. “imitation is suicide”

20. Life hands us a lemon, make lemonade

21. Forget our own unhappiness by trying to create happiness for others.

22. Be spiritual

23. Unjust criticism is often a disguised compliment. Remember that no one ever kicks a dead dog.

24. Do the very best you can, and then put up your old umbrella and keep the rain of criticism from running down back of your neck

25. Keep a record of fool things we have done and criticize ourselves. Since we can’t hope to be perfect, let’s do what E.H. little did: let’s ask for unbiased, helpful, constructive criticism.

26. Avoid thinking of others comments as trying to hurt you

27. Try not to think too much about one thing

**Getting things done (?)**

Know when something is adequate and get it done.

**Document extension:** emotional-Intelligence.docx,

**Work:**

1. Sincerity, ethics and honesty, Diligence, work 80 hours a week, enjoy what you do
2. Systems issues for setup: OpenMP runtime, moab scripts, compiler setup, .bashrc, ssh guide, machine access/banks, emacs editing, ssh key, macports, ‘use’ rose, use adept-utils
3. Baseline data: slack data histogram, noise histogram, increasing slack as we scale, dequeue overheads, other sources of noise (e.g. software error correction).
4. Develop theoretical analysis: Perf. Model based on baseline, engineering sched. parameters, execution time, validation of model with basic results, cost model.
5. Basic code/Implementation for dot product (make sure all optimizations finished, need to apply theoretical analysis here):
   1. Fortran:
   2. C:
6. App programmer usage + techniques: ROSE source-to-source, documentation of what the programmer has to do, use cases/corner cases when it doesn’t work, counting the number of MPI+OpenMP regions
7. CS knowledge:
8. Slack-conscious sched runtime: vectors, libunwind, wrap.py, slack pred strategies, overhead, error, noise/dequeue measurements, (optional: use additional slackpred methods)
9. Cmake software management: linking shared libs, high-precision timers, libraries (ssl, unwind, math), software arch.
10. Git: git stash, know commands, understand what’s going on, understand branches
11. Research Questions to setup experimentation:
12. Data points and .dat files org:
13. Data processing scripts/code: use python arrays, can use code from C here as well
14. Plotting scripts: .py scripts, organize plots
15. Processing results text output file of app run:
16. Apply techniques to actual Apps:
    1. NAS LU: ensure makefiles complete, update code
    2. AMG: run the amg files
    3. PF3D: run .i files
17. Performance results validation: Graph on varying sf, varying slack, varying noise, varying task sizes
18. Profiling to Explain Why it works: Costs of idle time and dequeue overhead
19. Persuade through intro: explain things one at a time, guide readers, state assumptions, “Elements of Style”
20. Literature search / related work: DPLASMA, Stanford work
21. Weekly Email updates: use good email etiquette, “elements of style”
22. Time management at work (schedules for projects, meetings, deadline): send out plan for year
23. Awareness of technology to be using in your field.
24. Marketing yourself and your work
25. Awareness of other people’s work/competition

Situations:

1. Ensure that the correct result is put in intro and abstract.
2. Latex:

- svn commit the .tex file with updated pdf generated from python script, and make sure to say something meaningful for it.

2. Python script:

a. Layout:

b. graph style and presentation: look up the best way to present the data (envisioning information)

speedup:

svn commit the script, svn commit the pdf generated and update the file in the latex as needed, making sure it looks correct.

make sure bash environment is setup correctly to have python from macports and pdflatex from macports

make sure you have the right .sty files and the correct paper format files

.dat file : put numbers in column

commit the .dat file

read numbers from the runtime, and print to .dat file: python script

scripts for running code: set environment variables, make sure it works for the particular machine in question

runFortranCodeVarySF, runFortranCodeVarySF.csh

ece110 / ece 210.

Specific Graphs:

1. Scalability Line graphs

- Make sure that lines don’t get squished, adjust scale

- check that all lines show

- Trends for each strategy on NUMA and non-NUMA .

1. Speedup graphs

CS Knowledge: Document extension for Code Practice: code-Practice.docx

Document extension for PCM overview: pcm-overview.docx

/ goal setting / (content)

**Work: mgmt: / admin: / travel: /comm:**

***Very long-term goals:*** My long-term goal within, say, the next 10 years is to use the research I have done during my PhD to make an impact to the field of computer science through offering new perspectives to a particular area of computer science or to develop an exemplar technological solution to a problem. (<HLFessay>).

***General Goal for Work***: Find a job at a company that does research. Write a paper this spring. Apply for a position to further distinguish yourself.

***Long-term goals (listen to what’s going on around you):*** Current state: HPC skills are important in a variety of fields. I am interested in continuing showing that my ideas are good, and developing my resume to be a leader in a company. I would like to make impacts for HPC in industry. I think a financial company is ideal.

***General Goal for Life:*** Better social int, friends base increase.

***Break down the goal:*** To do this, … I will need to … Read cracking coding interview, review webpages on questions. Do coding practice problems + Prepare behavioral questions. Mock interviews + learn from experience. Apply/Network. ***Notes on applications:*** Job postings, deadlines, application methods, hiring priorities, application documents, organization research, reference information, follow-ups, application status, requests for add’l materials, timelines, networking contacts, interview prep, post-interview notes, contact info, email initiation.

***Planning and logistics and politics*** … I plan to contact Blaise to ensure I get consideration. I will consider doing an IWOMP paper. I need to continue to support OpenACC work to get references. ***Upcoming deadlines:*** Argonne Training, Fulbright.

***Future work***: I want to get a job at Facebook. I then want to formalize the ideas. I hope that I can apply and formalize the ideas to other areas such as cloud computing. I then want to formalize the ideas to be societal. Media labs. To do this: … need to apply to jobs, Talk to people. Other directions (if the above doesn’t work):

1. Go to google/linkedin. Keep the papers going. Justify for why this works: This will allow me to maintain strength in what I do.

2. Go to media lab. Justification for why this works.

***Title:*** HPC rockstar. ***Casual Title:*** Software Consultant

***Work bios:*** tell people at work what you’re doing.

***Casual bio:*** (so, what do you do?): Software.

***Cover Letter***: Please tell why you would be a good fit for the position.

***Resume: Statement of Research: Publications List:***

***Reference Letters:*** Prof Cappello: / Prof Gropp: / Prof Johnson: / Prof. Garzaran: / Reference Letter by Prof Grigori

***Emails***: message to Blaise, message to Sandia, talk to Amanda,

***Taxes: / Admin/forms:***

***Travel planning:***

Strengthening: Building a network within work (think about how to do this, and how much priority this has): business industry, post-doc community, hpc community, tech industry.

Life:

Build social networks : To meet someone, I will need to increase fb posts for birthdays. I’ll need to add pictures every once in a while. I’ll need to show likes for news. I also need increase my presence online. I need to seek out events and plan out for the summer.

Find a girl:

- Meet through matrimony:

- Building a network outside of work (think about how to do this, and how much priority this has): facebook, SF events, weddings, Chicago events, PC/mgmt: Become better at work-life.

Document extension: resume-VivekKale.docx

Document extension: CoverLetter.docx.

Document extension for travel/exp: aachenTravel.docx

/ prio: / organization / mortgenson

**Spaces:**

***Jean Pockets:***

* Left pocket: Keys (RSA,car,house), cell phone (charger if needed) – unless it’s charging then remember to take it – Treat phone your baby !!
* Right pocket: wallet (40 dollars, credit card, driver's license, school id) - treat your wallet like its your baby 2 !!!

Vulnerability: right after coming home from car, taking off jeans for the night, at a club.

***Memorize:*** license plate num (E61 2444), ccnum, where you parked, add to sheet: miles per week = 100-150, gas per week = $30.

***Backpack***:

* main compartment: computer case (computer🡪 , computer charger)
* side compartment: keys, checkbook, pens
* left side outer: usb charger, outlet
* right side outer: lotion
* right side inner: glasses
* left side inner: badge

Vulnerability: don’t put on floor of hotel or on bed/couch.

***Office:***

* Left side: Fan, clock, phone, to file
* Right side: Food, tissue, papers, pens, hand sanitizer, tissue
* Cabinet 1: Right: work docs + books, Left: life docs + books,
* Center: Toiletries
* Cabinet 2: Papers
* Cabinet 3: Sleeping bag, pillow, misc., food
* Center: Computer 🡪
* Desktop:
* Docs:
* desktop background

Vulnerabilities: whenever I update the work docs.

***Car***:

* Glove compartment: insurance card.
* Trunk: water, clif bars, shorts.
* center console: earpiece, watch, sunglasses.
* passenger underseat: maintenance papers.
* driver underseat: checkbooks, receipts.
* windshield wipers, windshield washer fluid.
* tires check, oil change, maint req’d light.
* gas, windshield cleaning.
* Interior cleaning: front-inside of windshield, front seats, floor mats, side windows, center console, instrument panels, rear-view mirror, inside door entry.
* Exterior cleaning: wheels, hood, front sides, front door panels, rear door panels, trunk+bumper, roof

Vulnerabilities: have a busy week of driving and forget to fill gas, use earpiece to talk on phone.

***Bathroom***:

* clean toilet bowl
* check clothes in towel bar
* watch for water on floor
* clean sink area
* check for hairs on bathtub

Vulnerabilities: whenever done with shower.

***Kitchen:***

* fridge:
* pantry:
* cabinets:
* take garbage out.
* clean /mop floor, clean countertops, watch for boiling water

Finish organization book, Julie Mortgenson’s organization

Doors locked when going out, garage door, check stove

***Appearance:***

* contacts / check for glasses
* make sure belt is on, clean shirt
* pant should be high

***Bedroom:***

* *closet: top shelves: sport shirts, undershirts, polos*
* *closet: bottom shelves: pajama, underwear, jeans*
* center: bed: pillow, pillow covers, bed sheet, blanket
* left of bed: bedstand: top drawer:
* right of bed: desk

***Facebook:***

FB birthdays: ‘Happy Birthday man’ to close guy friends. ‘Happy Birthday <name>’ to girls. ,

***LinkedIn,***

***Laptop:***

1. Hobbies/tennis/bmw/jogging/humour-attention grabber.
2. Cover Letter.
3. Business Card.
4. Resume/CV, picture.
5. Other companies/ questionnaire.
6. Long-term goals.
7. Transcripts, Courses, Coding .
8. Statement of Research Interests.
9. Code examples, discussion of library, Research Experience
10. Publications, Awards and Honors, Conferences Attended, behavioral questions
11. Experience Excel Sheet.
12. Networking/LinkedIn.

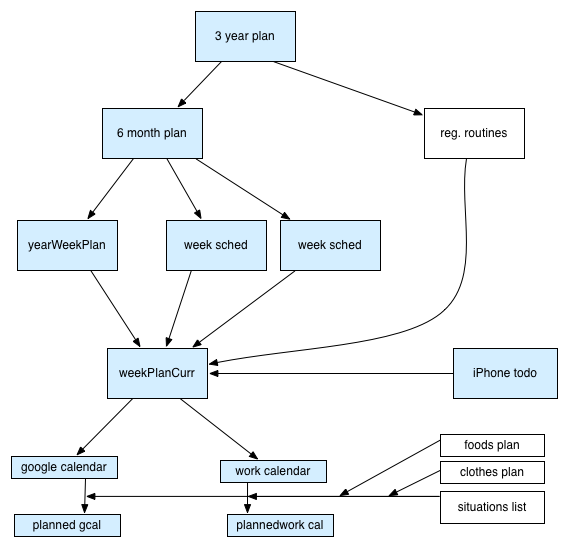
Girls – Relations

Make a daily plan to handle these things:

jean pockets (KeyWalletCell), Backpack (no crumbs, organize all pockets), office (clean desk of papers, get rid of hairs), laptop (clean outside, org folders), room (make bed, fold laundry, make sure no papers lying around), car (clean outside, get rid of crumbs, clean inside windshield), kitchen(milk in fridge, watch boil water), bath(water on floor, toilet bowl, take clothes out), lock doors, wash hands before/after eating something

When to throw things out:, Where to place certain things:

**Time:**



3 year plan:

Year week plan: Project goals list, Life goals list, monthly plan, Week-by-week plan .

Project todo list:

iPhone todo list: use labels Mgmt, Comm, Rel., add on

Weekly Mgmt: Weekly to do (look at items from projects, from iPhone, from gchat), Making time Estimates, Prioritization, Schedule

Regular Routines:

*Every Morning:* Oral Hygiene (5 mins), Brush Teeth**,** Mouthwash**,** Clean tongue - it helps with breath **,** Eating small breakfast**,** Check your schedule for meetings / appointments for that day: Is anything urgent? 5 mins**,** Excercise: 1 hour (Stretch for 10 mins, Treadmill for 30 mins , Lift weights or Yoga 1 hour, Meditation for 10 mins)**,** Shave: 5 mins (at least every other day) **,** Shower: 10 mins (shampoo+conditioner, acne wash,  soap on underarms/underbody) , dry hair, dry face, arms, legs, back, stomach, Clothing 5 mins to 10 mins**,** Comb hair, hair gel optional:(1 min.)**,** Remember to lock door, Keys/Wallet/Cell/Badge (get a bucket) (1 min.)**,** Any required medicines (1 min)**,** Wear lenses and rinse out case (2 mins)**,** Deodorant or cologne depending on the day 1 min**,** Wash your face with a face wash - it helps with acne (3 mins.)**,** Moisturize - apply lotion  (2 min)**,** Neti pot / clean out sinuses - if you’re feeling congested:5 mins **,** clean out ears, nose: 5 mins.

*Every Evening / Night:* Call family: 30 mins**,** meet friends: 20 mins, Cleaning dishes / cleaning house: 10 mins, Lay out clothes for the next day, iron if necessary, start laundry if needed: 10 mins, Collect anything you’ll need for the next day (e.g. dry cleaning to drop off): 5 mins, Check Facebook/linkedin/google+: 10 mins, matrimony stuff: 10 mins, Food for lunch/dinner the next day: 10 mins, Situation analysis: 10 mins

*Going to Bed:* Finish computer stuff: 5 mins ***,*** Get into sleeping clothes: 1 mins, Before sleeping: brush teeth, floss - 2 mins ***,*** Properly wash and put away contacts: 2 mins***,*** Review the days good things: 10 mins ***,*** Light Reading: 10 mins, Prayer: 10 mins

Sat: analyze past week, spaces org, clean room,

Sun: long-term planning, week plan, running,

Mon: meeting notes, coding , prep for Toastmasters.

Tuesday: Weights, Toastmasters @ 12PM.

Wednesday: Weights .

Thursday: Yoga

Friday: Weights

Clothes planning: coordination/Appearance details/ matching/ what to wear when: shirts, pants, underwear, socks, have combos set up, put them on calen

Foods planning: Groceries (milk, juice, bread, fruits, frozen foods), Dinner (spaghetti, indian, Mexican, Burger, Burrito

Calendar:

* every year, put new routines on calendar
* every month, put upcoming big travel plans
* every week, update calendar
* every day, look at calendar for the next day, and see which situations arise.

Identify situations and put them up on calendar, and then put the situations pre-notes and post-notes to the calendar.

Think 20 mins ahead to see what’s happening.

Think a day ahead of time what’s happening and plan ahead .

Gen Knowledge: Directions/Logistics/Driving/Car

Gen. Knowledge: weather, technology, news.

Dinner Etiquette:

Health/Gym:

Budget/Money:

**8th habit:**

**Public victory:**

**MAPS:**

Habits summary:

Add other notes on structuring your thoughts here.

Introduction:

Point 1: subpoint 1, subpoint 2, subpoint 3, Point 2: subpoint 1, subpoint 2, subpoint 3; Point 3: subpoint 1, subpoint 2, subpoint 3.

Conclusion:

Done when …

**Elements of Style: (TODO: add on details): 20 minutes**

Habits: Win/win : -> small loose connectors

Habits summary: win/win:

TM: how to say it:

Introduction:

*I. Grammar: e.g., when to use semi-colon, e.g., using sound good rather than sounds good.*

1. place comma before conjunction introducing independent clause.

3. use proper case of a pronoun.

4. participial phrase must refer to the grammatical subject.

*II. Principles of Composition: e.g., put emphatic words at the end.*

1. choose a suitable structure and hold to it.

- real-world example: consider structure of intro.

2. make the paragraph the unit of composition:

- see paragraphs in paper.

- break paragraph into two parts only if needed.

3. Use active voice:

a. I shall always remember my first visit to Boston vs. my first visit to Boston will always be remembered by me.

b. Avoid “there were”, “It was”, “It is”.

Real-world examples: look at writing in paper.

4. Put statements in positive form:

a. avoid the word “not”.

b. avoid indefinite language.

c. place a negative and positive in opposition makes a stronger structure.

d. Don’t use words like “would”, “should”, “could”, “may”, “might” and “can”. These can show uncertainty / Real-world examples: consider text messages to Nithya.

1. Use definite, specific, concrete language.
2. Omit needless words.
3. Avoid a succession of loose sentences.
4. Express coordinate ideas in similar form.
5. Keep related words together.
6. In summaries, keep to one tense.
7. Place emphatic words of a sentence at the end.

*III. Matters of form:*

1. Colloquoialisms.
2. Avoid exclamation.
3. Headings.
4. Margins.
5. Numerals.
6. Parenthesis.
7. Quotations.
8. References.
9. Syllabication: when a word must be split across a line, consider a dictionary to see how word should be split.
10. Titles.

IV. Commonly Misused words, e.g., that vs. which. Vis. Through word network.

1. Aggravate. Irritate.
2. All right.
3. Allude.
4. Allusion.
5. Alternate. Alternative.
6. Among. Between.
7. And/or.
8. Anticipate.
9. Anybody.
10. Anyone.
11. As good or better than.
12. As to whether. Whether is sufficient.
13. As yet. Yet is nearly as good if not better.
14. Being. Not appropriate after regard … as
15. But. Unnecessary after doubt and help.
16. Can.
17. Care less.
18. Case.
19. Certainly.
20. Character.
21. Claim.
22. Clever. A clever horse is a good-natured one.
23. Compare.
24. Comprise.
25. Consider.
26. Contact.
27. Cope.
28. Currently.
29. Data.
30. Different than.
31. Disinterested.
32. Divided into.
33. Due to.
34. Each and every one.
35. Effect.
36. Enormity.
37. Enthused. She was enthused about her new car. She enthused about her new car.
38. Etc.
39. Fact.
40. Facility.
41. Factor.
42. Further. Farther.
43. Feature.
44. Finalize.
45. Fix.
46. Flammable.
47. Folk.
48. Fortuitous.
49. Get.
50. Gratuitous. Unearned or unwarranted.
51. He is a man who.
52. Hopefully.
53. However.
54. Imply. Infer.
55. Importantly.
56. In regard to. As regards.
57. In the last analysis.
58. Inside of. Inside.
59. Insightful.
60. In terms of.
61. Interesting.
62. Irregardless.
63. –ize
64. kind of.
65. Lay.
66. Leave.
67. Less.
68. Like.
69. Line. Along these lines.
70. Literal. Literally.
71. Loan.
72. Meaningful.
73. Memento.
74. Most.
75. Nature.
76. Nauseous. Nauseated.
77. Nice.
78. Nor.
79. Noun used as a verb.
80. Offputting. Ongoing.
81. One. One of the most.
82. –oriented.
83. Partially.
84. Participle for verbal noun.
85. People.
86. Personalize.
87. Personally.
88. Possess.
89. Presently.
90. Prestigious
91. Refer.
92. Regretful.
93. Relate.
94. Respective. Respectively.
95. Secondly, thirdly, etc.
96. Shall. Will.
97. So.
98. Sort of.
99. Split infinitive.
100. State.
101. Student Body.
102. Than.
103. That vs. Which.
104. The forseeable future.
105. The truth is.. the fact is
106. They. He or she.
107. This.
108. Thrust.
109. Tortuous. Torturous.
110. Transpire.
111. Try.
112. Type.
113. Unique.
114. Utilize.
115. Verbal.
116. Very.
117. While.
118. –wise.
119. Wise.
120. Worth while.
121. Would.

V. General principles, e.g., write from the heart, have sympathy for reader:

1. Omit needless words, avoid a succession of loose sentences, express coordinate ideas in similar form, keep related words together, keep to one tense in summaries(know what tense to use), use concrete and specific language, emphatic words go at the end of a sentence.

Style documentExtension: StyleGuide.docx

**How to Win Friends and Influence People** - 30 minutes.

Public victory, Habits:listen, syn - TM: 8 , emot: empathy (shows connection to emot intelligence).

Introduction:

* Never criticize, condemn or complain.
  + Two-gun Crowley: people don’t believe they’ve done anything bad. In the extreme case, a criminal doesn’t believe he/she has done anything bad.
  + Regular people like managers don’t criticize. Introspecting Wanamaker himself says he had trouble overcoming/fretting about his own limitations of intelligence. When he thought about himself, he realized what it meant for others.
  + Animal learns from reward rather than through punishment.
  + Father forgets: habit of finding fault is bad. I think you are good as you are.
* Give honest and sincere appreciation.
  + Only one way to get anybody to do something: make the other person want to do it.
  + Freud: Everything you and I do springs from two things: sex urge and desire to be great.
  + Many things are needed, but the desire to be great is something that is seldom gratified. The desire to be great is chief in distinguishing between humans and mammals. Hogs didn’t care about the ribbons they won, but father did. If ancestors didn’t want importance, then civilization would be impossible.
  + How you get your feeling of importance 🡪 tells about what you are. Dillinger and Rockefeller.
  + Mannerisms: George Washington wanted to be called “mighty”. Wife of Lincoln said how dare you come to my presence until I invite you.
  + People become invalids to get attention and be important. Woman took to her bed, and for 10 years the mother went to help her everyday. One day the mother died. For some days, the girl was sad. Then, she got up and did her own things.
  + Consider Fictitious story of a woman putting hay in front of men. The day she did that, the men noticed. For years before, none of them noticed the good in her cooking.
  + Study was made on runaway wives, the number one reason was “lack of appreciation” -🡪 not true today …
  + Difference between flattery and appreciation:
    - flatter is insincere and appreciation is sincere.
    - Flattery comes from teeth out, and the other comes from the heart out.
    - Appreciation is unselfish and Flattery is selfish.
    - One is universally admired and the other is universally condemned.
  + “Don’t be afraid of enemies who attack you. Be afraid of the friends who flatter you.”
  + Flattery is Cheap praise. Flattery is telling the other person precisely what he/she thinks about himself. Ralph Waldo Emerson: you can never say anything but what you are.”
  + We spend 95% time thinking about ourselves. Stop to think extra about good points, we don’t need to resort to flattery.
  + Hurting people doesn’t change them; it is never called for. I pass this way only once. Let me show kindness now. Later on, I may not be able to do so.
  + Emerson said “All men are superior to me – I learn from them. “ If Emerson said that, then it’s a 1000 times more true for you and I.
  + Give hearty approprabation and people will remember it years later.
* Arouse in the other person an eager want.
  + Dale Telling kids to not break rules, by coaxing them to do something.

Make People like you:

* Smile.
* Become genuinely interested in other people.
* A person’s name is the most important sound in any language.
* Be a good listener. Encourage others to talk about themselves.
* Talk in terms of the other person’s interests.
* Make the other person feel important, and do it sincerely.

Listen

1. Avoid argument (Saturday, feb 1st)
2. Show respect for the other person’s opinions. Never say you’re wrong. (Saturday, feb 1st)
3. If you are wrong, admit it quickly and emphatically
4. Begin in a friendly way
5. Get the other person saying “yes, yes” immediately before starting the conversation.
6. Let the other person do a great deal of talking.
7. Make the other person feel that the idea is her own
8. Try to honestly see things from the other person’s point of view .
9. Be sympathetic to the other person’s ideas and desires
10. People have two reasons for doing something: sounds good and a real one
11. Dramatize your ideas: (merely stating the truth isn’t enough, but truth has to be made vivid, interesting and dramatic).
12. Begin with praise and honest appreciation.

Leadership:

* + - 1. How to criticize – and not be hated for it.
      2. Talk about your own mistakes first.
      3. No one likes to take orders.
      4. Let the other person save face.
      5. How to spur people on to success
      6. Give a dog a good name.
      7. Make the fault seem easy to correct.
      8. Make people glad to do what you want.

WinFriendsInf: Document Extension: winFriendsInf.docx

**Conversation Skills:**

(look at video) -

words to use – you, we,

**TM: 30 minutes:**

(write down intro)

General evaluator:

Timer/grammarian.

Speaker.

Speech evaluator. Table topics presenter, table topics answering.

**Social Intelligence**: add basic summary – 1 hour.

Habits: synergize -- TM: 10: experiences: ( ece210 ) habits summary :

30 minutes every day –passage reading - read (5 minute), make notes (2 mins ), write an example (3 minutes), connect with other related topics, integrate into wl.

SocialInt: Document Extension: socialIntAndTM.docx.

Working with emotional intelligence. (Ece 329). iPhone.

Sharpen the saw:

evaluations:

facebook, linkedin, Microsoft,

(Organize a bit better – 30 mins) :

***Matrimony/Relationship Mgmt :***

Picture1 , Picture 2

Later (R1438138) 

**A few words about my son:**

My son grew up in the US. He is a handsome and athletic person. He is hardworking and studious. In his free time, he enjoys playing piano. He also plays and follows tennis. He is doing a PhD in Computer Science, and has a fellowship to work in a government laboratory while doing it. **Basic Details:** Name: Later, Body Type / Complexion: Athletic / Wheatish, Age: 29 Yrs, Physical Status: Normal, Height: 6 Ft / 183 Cms, Weight: 76 Kgs / 167 lbs, Mother Tongue: Marathi, Marital Status: Never married, Eating Habits: Eggetarian (vegetarian), Drinking Habits: Light / Social drinker, Smoking Habits: Non-smoker **Religious Information:** Religion: Hindu, Caste / Sub Caste: Brahmin Deshastha (Caste No Bar) / Rigvedi, Gothram: Vishwamitra, Star / Raasi: Not Specified, Manglik: Don't know **Location:** Country: United States of America, City: Berkeley, State: California, Resident Status: Citizen, Citizenship: United States of America **Professional Information:** Education: MSc IT / Computer Science, Education in Detail: BS Computer Science, then worked for a while before returning to higher studies, Occupation: Student, Occupation in Detail: PhD student/ Fellowship, Employed in: Government/PSU, Annual Income: Not Specified **Hobbies & Interests:** Hobbies: playing piano, Interests: Politics, Health & fitness, Favorite Music: Hip-Hop, Rap, Techno, Jazz, Western classical, Pop, Sports/Fitness Activities: Cycling, Tennis, Jogging / walking, Weight lifting, soccer, Favorite Cuisine: South Indian, Thai, Punjabi, Italian, Preferred Dress Style: Casual wear, Western formal wear, Spoken Languages: English, Marathi

**Family Details:** Family Values: Moderate, Father's Status: Professor, Family Type: Nuclear, Mother's Status: homemaker, Family Status: Upper middle class, No of Brother(s): 1 - Not Married, Ancestral Origin: Not Specified, No of Sister(s): None **About our family:** We are a close-knit family. We have been in the US for the last 30 years. His younger brother works in Chicago. We are well-connected with India, with siblings in Pune.

**PARTNER PREFERENCE**

**Basic & Religious Preferences**

Brides's Age: 24 – 30 Yrs, Height: 5 Ft - 6 Ft / 152 Cms - 183 Cms , Physical Status: Normal, Eating Habits: Doesn't matter, Smoking Habits: Non-smoker, Drinking Habits: Doesn't matter, Religion: Hindu, Mother Tongue: Any Mother Tongue, Caste: Any Caste, Sub Caste: Any Sub Caste, Gothram: All (Except my gothra), Star: Any Star, Manglik: Doesn't matter**,** Education: Any Degree, Occupation: Any Occupation, Annual Income: Any Income**,** Citizenship: Any Citizenship, Country: United States of America, (India), Residing State: Any, Any, Residing City: Any City

**What we are looking for**

|  |
| --- |
| Must be willing to live in the US. Prefer someone who spent many years in the USA.  We hope to find someone who has a happy disposition and understanding nature. |

*Rules for parents involvement:*

1. Every on Sunday, both Vivek and mom and dad go through the people and filter those that are likely candidates.
2. Have a call with me to discuss “yes”, “no”, “maybe”. For those that are “yes”, respond within two weeks. For those that are “no”, respond giving one of the stock responses(“looking for U.S. citizens” , “not what we are looking for”).
3. If pictures are asked for, give only 3 additional pictures. These should be the same for each person that requests.
4. In the initial interest by parents, do not try to sell yourself and me at all. Just answer the questions that are asked. They can already google things about you to figure out whether you are a “Professor at the University of Illinois”.
5. When the mother is talking, please have mom talk to her directly. I don’t want dad talking to the mother.
6. Also, I want mom’s signature to be on some of the emails.
7. When asked about specific information about me, such as “what I’m doing , plans for the future”, say vehemently that “Vivek will tell more about this”
8. Before getting involved in a phone call, please let me know about it!
9. Before starting an email exchange (even if they need a response quickly), please let me know about it! Let me know about specific questions you might have.

*Stock Responses:*

*Interests:*

*Mgmt:*

- Every weekend, set aside some time for responding.

- Every night, check new interests and add on to list.

- talk to parents

Dating guide:

* Don’t talk about past relationships
* Know when to hold hands, when to show affection, when to kiss
* Listen carefully to conversation
* Don’t say anything negative about girl, even if she says negative to you.
* Stay calm in tough situations
* Be happy and smile, act like you are having fun
* Add humour every now and then
* Connecting/find common ground
* Understand married life
* Send flowers to show you care, when interested.
* Be intentional with what you are saying, make sure you organize your thoughts
* Shout to look confident
* Avoid talking too long on phone, keep dates short and simple
* Remember that a girl’s mind can’t forget things often

Channels :

* Match
* Matrimony
* Eharmony
* website extension: experiences
* CoffeeMeetsBagel

DocumentExt: Experiences.doc

Attic:

Resume + applications

Dos and dont’s: put everything in central spot.

Spreadsheet for jobs, Job search apps, Paper? , Project management software, Make a schedule: work for 30 minutes a day: , Make templates: , Mgmt: rel: vulnerability: Back it up: (use box). Todo: next steps.

To do: next steps: what are 3 things you can do in the next week to get on track to stay organized?

1. Work: admin: Create a word document with one full application cycle.
2. Work: admin: Create an excel spreadsheet to store the different apps.
3. Work: admin: Read interview book to get additional information to shape. And talk to go.illinois.edu/GradCareersAppt/

Purpose:

1. provide a runtime to automatically adjust scheduler parameters for obtaining further performance improvements of application codes.

2. substantially reduce application programmer effort to use our technique through automatic transformation of the application programmer’s code. Functionality:

Introduction: ---

Your research:

Schedule:

Saturday: review all code/problems.

Sunday: put together and rehearse.

Step 0: know material ?

Step 1: Resolve ambiguity

Step 2: Design an algorithm

Different ways to solve:

Examplify(angles in clock), Pattern Matching, Simplify and generalize, base case and build, data structures

Step 3: pseudocode -

Step 4: Write code - computing: correct, clear,

Step 5: Test code and carefully fix mistakes

Step 6: Analyze:

Examplify, Pattern Matching, Simplify and Generalize, Base case and Build, Data Structure Brainstorm

**Schedule:**

Logistics: Calmness. Back straight, no body noises/farts, speak clearly with same pitch and speak loudly. Long-term goals + Behaviorial questions: , Your research:

Discuss project in second on Jacobi -

Answer the following questions: Project size , code size.

Questions:

Follow formula:

Assumptions: /Ask Questions

Idea: < go through 5 methods of approaching> .

Code: <coding standards>

Test / prove correctness/ find test cases: avoid quickly changing algo

Analysis:

Probability and combinatorics:

Numerical algorithms

NP-complete

Knapsack

TSP

Randomized / Probabilistic algorithms / Dynamic programming

Algorithm analysis

Algorithms: sorting, searching

Heaps and Priority Queues

Graphs and Trees

Tries

Stacks and Queues

Linked lists

Hash tables

Arrays - algorithms, trees,

Pointers

Watch for top 10 mistakes

Top 10 mistakes: practicing on a computer, not rehearsing behavioral q’s , not doing a mock interview, trying to memorize solutions, not solving problems, rushing , sloppy coding, not testing, fixing mistakes carelessly, giving up.

Question:

Willing to relocate/ Start date.

Companies:

Questions for interviewer:

Thank you.

successful software to learn from and to consider

Algs

Algs + Data structures

C++/style/emacs

Patterns / Arch.

Signals

**Behavioral Questions:**

Questions to ask NVIDIA:

1. What is the new initiative on automation in cars I saw in the news?

Nugget: Let me tell you about a time when I had to work in a team of four people.

Situation: I worked with four supervisors. Each were experts in their field and senior. Several pulls in the group.

Action: I reduced the size. I made collaborative notes before each meeting, and pointed out issues to be taken care of. I also found latex useful rather than assigning tasks on git.

Result: fast responses. People had more targeted advice/input rather than general statements. I got more things done and had time to spare.

What it says: ability to solve problems in a group setting. Ability to work with senior people/ experts .

**Basics Notes:**

Follow formula.

Combinatorics:

Probability - 1 day

Numerical algorithms:

NP-complete:

Knapsack/TSP:

Randomized / Probabilistic algorithms / Dynamic programming

Algorithm analysis:

Algorithms: sorting, searching

Heaps and Priority Queues

Graphs and Trees

Tries

Stacks and Queues

Linked lists

Hash tables

Arrays

Pointers / oop

Software

Ah ok.

It was nice outside though right?

Top 10 mistakes (vulnerabilities):

Schedule:

* Practice recursion problems: - 1 hour
* Practice with implementation for each of the different problem categories:
  + 2 hours
* Review basics from 1 and 2: - 2 hours
* Document in notes: - 2 hours
* Add things to memorize : 1 hour

**BrainTeasers / Probability:**

**Mathematics:**

Sieve of erasthosines:

Note that a prime number is a number whose only divisor is 1 and itself. A composite, or non-prime, number has a factorization of a set prime numbers. 1 is neither prime nor composite.   
  
**Question: Design an algorithm to find the kth number such that the only prime factors are 3, 5, 7.**   
  
To help clarify this question:  
  
- Generally, the algorithm needs to find the kth number from the sequence of numbers (increasing order) having the property that the only prime factors are any subset of the set {3, 5, 7}.

- Example: The fifth such number is 3\*5 = 15.   
  
-------------

Solution:

Assumptions: The kth such number is positive and an integer. The number k is greater than 0.  
  
Idea:   
  
Enumerate the first several numbers in the sequence of numbers and hope to see a pattern.

Thinking conceptually/intuitively: The first such number is 3. The second such number is 5. The third such number is 7. To get the next number, you need to multiply current number by 3. To get the next to next number, you need to multiply the next number by 5. And so on.

Considering the definition of primes, the sequence enumerates in ascending order the numbers that are of the form (3^a)\*(5^b)\*(7^c).   
   
3, 5, 7, 3\*3, 3\*5, 3\*7, 5\*7, 3\*5\*7,3\*3\*7, 3\*3\*5\*7, 3\*5\*5\*7, 3\*5\*7\*7, 3\*3\*5\*5\*7, 3\*5\*5\*7\*7.   
  
The first such number is 3. The second such number is 5. The third such number is 7. To obtain the next number, you need to multiply current number by each of 3 , 5 and 7 and obtain the smallest of the three products.   
 f(0) = 1   f(1) = 3   f(2) = 5   f(3) = 7.

The general pattern is that the kth number is min{3\*(k-1)th number, 5\*(k-1)th number, 7\*(k-1)th number }   
  int f(int num\_k)                                                                         
    {                                                                                                                                                                    
      if(num\_k < 0)                                   
        printf("error: value can't go below zero");                                           
        exit(1);                                                
      if (num\_k == 0)    return 1;                                                                             
      else if (num\_k == 1)  return 3;                                                                             
      else if (num\_k == 2)  return 5;                                                                             
      else if (num\_k == 3)  return 7;                                                                             
      else                                                                                         
          // Note that min functions are associative  , so can do the below.                    
          return min(3\*f(num\_k -1), min(5\*f(num\_k - 1), 7\*f(num\_k -1)));                     
    }

Test cases:

int main (int argc, char\*\* argv)   
{assert (f(5) , 15); f(100); cout << f(23);}

Powers of 3:

**Computability Theory:**

Memorize:

Exponential: 3-SAT

NP-hard: / NP-complete: Knapsack, Traveling Salesman Problem

Polynomial: sorting

Decidable, recognizable

Turing machine

CFG

FSM

What is an algorithm?

What is time?

Al-Khwarizimi - algorithm started from numerical linear quadrature :

Program 🡨🡪 algorithm

Programming language 🡨🡪 pseudocode/ structured English

Computer 🡨🡪 model of computation

Model of computation Specifies 1. What operations an algorithm is allowed

2. Cost (time, ) of each operation

Random Access Machine (RAM)

Modeled by big array . Giant array - 4 gb . access in constant time

In Theta(1) time, an algorithm can

Read in or Load O(1) words

Do a constant number , O(1), computations on them

Store O(1) words.

O(1) registers

What is a word? A word is w bits. Manipulate words.

2. Pointer machine: dynamically allocated objects. Object has a constant number of fields. A field is either a word or a pointer.

Can implement model in random access machine.

Pointer Machine

**Python:**

**Revise edit distance from Harvard class**

Document = sequence of words .

**Word = string of alphanumeric**

- document = sequence of words

- word = string of alphanumeric

- idea: shared words

- think of document as a vector

- d[w] = of occurrences of w in d. distance between two vector

d(D1, D2) = D1\*D2.

Divide by the length of the vectors.

Angle between two vectors: D1\*D2 / |D1|\*|D2|

-vector : x-axis is words, y-axis is number

* d’ (D1, D2)
* d1 = “the cat” , “the dog”
* split document into words.
* copute frequencies of words
* for (word in doc)
  + count[word] += 1;
  + O(word1);
* dot product .
* runtimes of different versions of algorithms: 1: 228.1 2: 164.1 3: 123.1 4: 71.7 5: 18.3 7: -- small changes make dramatic differences in performance.

Python model:

1. “list” = array

e.g., L[i] = L[j] + 5;

2. object with O(1) attributes

x = x.next ;

L.append(x); //table doubling - Lecture 9 : can be done in constant time .

L = L1+L2 ;

L = [] ; for x in L1: L.append(x)

For (x in L) - linear

Len(L) - counter built-in

For x in L1:

L.append(x)

For(x in L2)

L.append(x)

L.sort() 🡪 O(L lg L) :

Dict: D[key] = val; 🡪 O(1) key in D // constant time with high probability.

Long: L 1

X +y O(x + y )

x\*y O(|x| + | y| ) ^lg3)

heapq: Lecture 4 – O(lgn)

Document distance problem : give two documents , compute distance between them.

Motivation: cataloguing the web : need to find distance difference

Wikipedia: mirrors, find document distance.

Catch two people cheating .

d(D1 , D2)

web search : “introduction to algorithms “ think of algorithms that are similar to other …. Test which document is most similar to introduction to algorithms

**Recursion:**

**Dynamic Programming:**

*Basics:*

Hear a problem like: Write code to list the first n ..

Content:

Recursive solutions are built off solutions to sub-problems.

Recursive soln: to compute f(n), you need to add,remove something or changing solution to f(n-1).

Bottom-up recursion: most intuitive. Solve problem for a simple case / for one element, like a list with one element, then solve problem for two element , then solve problem for three elements, and so on. Key is to Build solution for one case off the previous case.

Top-down recursion: more complex. Think about how to divide problem for case N into subproblems . Be careful about overlap between cases.

Example: Fibonacci. (memorize)

int Fibonacci (int i) {

if(i == 0) return 0;

if(i = 1) return 1;

return fibonacci(i -1) + fibonacci(i -2);

}

int[] fib = new int[MAX\_INT];

int fibonacci(int) {

if(i == 0) return 0;

if (i == 1) return 1;

if(fib[i] != 0) return fib[i];

fib[i] = fibonacci(i-1) + fibonacci(i-2);

}

Recursive vs. Iterative: recursive can be space inefficient. Each recursive call adds new layer to stack.

=======

*Problem:* A child is climbing n steps. The child can climb either 1 stair, 2 stairs or 3 stairs at once. Implement the number of ways a child will climb the stairs.

*Idea:* Total number of ways is the number of ways of reaching n -3 step to nth plus number of ways of where the last step is 1, the number of ways of where the last step is 2 hops , the number of ways for n-3 where the last step is 3 hops.

Climbing n-3 steps and climbing 3 steps

Number of ways climbing n-2 steps and then climbing 2 steps.

Number of ways climbing n-1 steps and then climbing 1 steps.

Top down decision tree :

*Code:*

public int countWays(int n) {

if (n < 0 )

return 0;

else if (n == 0)

return 1; // the one step to account for each of the three possibilities.

else

return countWays(n-1) + countWays(n-2) + countWays(n-3) ;

}

public static int countWaysDP(int n, int[] map){

if (n < 0) return 0;

else if (n == 0) return 1;

else if (map[n] > -1) return map[n];

else

map[n] = countWaysDP(n-1, map) + countWaysDP(n-2, map) + countWaysDP(n-3, map);

}

*Analysis:* 3^n

*Tests:*

Problem: all pairs that add up to specified sum.

Problem: document distance.

Problem: largest with black border.

Go through all subsquares with top-left index i,j starting with largest subsquare, checks if it has black border. If not, it moves to the next smallest and checks black border. Continue until it finds a square with a black border(this is a subroutine. We store the submatrix size in array B[i\*arrSize+j]. We then loop through this array to find the maximum black border.

**Problem 1**: Prefix sum:

Idea:

b[0] = a[0];

for (i = 1; i< n; i++)

b[i] = a[i-1] +b[i];

**Problem 2:** Find missing number in array

Idea: if the difference between two consecutive elements is greater than 1, the element is missing.

Code:

if(a[0] +1) > 1)

return 0;

for(i =1; i< n; i++)

if (a[i] – a[i-1])

return i;

**Problem 2:** Water

**Problem 11:** find the subsequence in an array with the maximum sum.

*Idea:* use a running best. At each step choose between the subsequence ending with a[i] or the longest subsequence.

void findSubSeqMaxSum(int\* a, int\*m, int\* k, int n, int\* sum)

{

int b = 0;

int rb =0;

int l = 0;

\*m = -1;

\*k = -1;

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

rb = rb +a[i];

if (rb < 0) {rb = 0; l = i +1;}

if (rb > b) { b = rb; \*m = l; \*k = i;}

}

\*sum = b;

}

**Problem 11:** find the longest increasing subsequence.

*Idea:* Calculate the length of each increasing subsequence starting from i. To find the increasing subsequence, keep iterating until you’ve decreased count. Keep track of subsequence that’s the longest as you progress through i.

**Problem 12:** given an array, sort the array into an alternating peaks and valleys.

Assumptions: positive integers, don’t need to return the index – just the number. At endpoints, assume that an element is a peak only if the right elt. is a peak.

Example: {5, 3, 1, 2, 3}, {5, 1, 3, 2, 3}.

Idea1: take elements of array, and add to secondary array ensuring that each time we add an element it satisfies the condition. The condition switches each time we add an element.

Idea2: Loop through each element until it satisfies the condition of

Idea3: Median

Pseudocode:

Code:

Analysis:

**Sorting and Searching: TODO:**

1. write down insertion sort and quick sort.
2. Note the solutions to problems from book.
3. Note the solutions to harder/leetcode problems.

Why sorting?

Obvious: phone book. Mp3 organizers, spreadsheets,

Problems that become easy: finding a median of a list

Searching: rendering.

Bubble sort/insertionsort/selectionsort, Quicksort, Mergesort, Radix sort:

Not so obvious apps: files compression, subroutine in data compression

Layers corresponding to scene - render front to back don’t have to worry about opaque.

Cocktail sort, bitonic sort.

Do a binary search on a[0, I -1] in O(lg i) time . O(nlgn);

Insertion sort:

for (i = 1, 2, .. n)

insert a[i] into sorted array a[0, i-1];

insert i into sorted array by pairwise swaps down to correct position;

Counting comparison in the sorting algorithms: O(n) steps in terms of key positions O(n2).

Operation is a compare and swap. What if compares are more expensive than swaps? What’s a simple fix to the algorithm where complexity?

TODO: check the conceptual understanding.

Do a binary search to do the swaps? O(nlgn) compares.

Key swap: 5, 2, 4, 6, 1, 3 🡪 2, 5, 4, 6, 1, 3 🡪 2, 4, 5, 6, 1, 3 🡪 2, 4, 5, 6, 1, 3 🡪 1, 2, 4, 5, 6, 3 🡪 1, 2, 3, 4, 5, 6

Divide and Conquer: Merge sort:

size n , 2 arrays of size n/2 ,

Merge sort: sort (A) --

L R

Sort Sort

L’ R’

L = [ 0 ,

Merge

Sorted array A

Merge two arrays as input

L’ R’

20 12

13 11

7 9

2 1

two finger algorithm : (!) 🡪

Analysis: O(n) -- how did he come up with this?

T(n) = c1 + 2T(n/2) + c\*n ; < -- proof generalizes for f(n) instead c .

Proof of analysis by picture -- expend recurrence out

Draw tree: Cn 🡪 c n/2 . Each level costs cn . Remember, Levels: 1+ lgn . So, total cost is (1+ lgn) \* cn

Advantage of insertion sort over merge sort . :

Merge sort: O(n) auxillary space . in-space sort 🡪 O(1) auxillary space.

Merge sort: in-place merge sort - impractical

2.2 n lg(n)

Insertion - python: O(0.2n2)

C: O(0.01n2)

Intuition on recurrence relations and master theorem : TODO: finish video.

// Approach 2: do a comparison, like you do for a max.

int\* insertionSort(int\* arr, int n)

{

int i,j;

int temp;

for(i = 1; i<n; i++)

{

temp = a[i];

j = i - 1;

}

while((temp < a[j]) && j >= 0)

{

a[j+1] = a[j];

j = j - 1;

}

a[j+1] = temp;

}

return a;

}

int\* selectionSort(int\* arr, int n)

{

int i,j;

for(j = 0; j < n-1; j++)

{

int iMin = j;

for(i=j+1; i<n; i++)

if(a[i] < a[iMin])

iMin = i;

if(iMin !=j) {

temp = a[j];

a[j] = a[iMin];

a[iMin] = a[j];

}

return a;

}

}

//Approach 1: randomly choose a pivot. Divide the array into two parts, that which is greater and that which is less.

// Test cases:

// Correctness:

// Analysis:

quicksort(int\* arr, int n)

{

int index = partition(arr, left, right);

if(left < index -1) quicksort(arr, left, index –i);

}

partition(int\* arr, int left, int right)

{

int pivot = arr[(left + right)/2];

while (left <= right)

{

while (arr[left] < pivot) left++;

}

}

// Approach 3:

mergeSort(int[] array, int[] helper, int low, int high)

{

int[] helper = new int[array.length];

}

mergeSort(int[] arry, int[] helper, int low, int high)

{

if (low < high)

{

int middle = (low + high)/2 ;

mergesort (array, helper, low, middle);

mergesort(array, helper, middle+1, high);

merge (array, helper, low, middle, high);

}

}

radix sort

- imagine each integer as base b.

- num digits = d - logb (k) + 1

* sort inints by least sig. digit

- sort ints by least sign. Digit

- sort using at most d .

// Problem: search for an element in an array.

// Solution: Binary search.

int binarySearch(int[] a, int x)

{

int low = 0;

int high = a.length -1 ;

int mid;

while(low <= high)

{

mid = (low + high) / 2 ;

if(a[mid] < x)

low = mid + 1;

else if(a[mid] < x)

high = mid – 1;

else

return mid;

}

return -1;

}

}

int binarySearchRecursive( )

{

if(low > high) return -1 ;

if (a[mid] < x)

return binarySearchRecursive(a, x, mid+1, high) ;

else if ( a[mid] > x)

return binarySearchRecursive(a, x, low, mid -1);

else

return mid;

}

alternatives : search a hash table, consider a binary tree

**Problem 3:** peak finding

Idea:

if (a[i] > a[i+1] && a[i] < a[i-1])

return i; // Note that boundaries don’t matter.

**Problem 2:** Merge array A and array B.

Idea: Compare elements A and B and inserting them in order, until we’ve exhausted all elements in A and in B. Don’t want shift existing elements backwards, so shift elements into the back of the array.

**Problem 3:** Sort an array of strings so that all anagrams are next to each other.

Assumptions: don’t need to sort array alphabetically, only group.

*Idea1:* Apply sorting algorithm to array of strings, and change the comparator. Comparator will be used to indicate that two strings which are anagrams of each other are equivalent. Count all occurrences of characters in two words and check if they match. Or, sort the characters in the string and check if the sorted strings match – do this. ( -- Question for clarification: How does changing the comparator make words that are anagrams near each other? If they are equal, then we don’t change order. This will also order anagrams alphabetically, but we don’t need to do this. --).

*Idea2:* Group words by anagram through a hash table. Run through the list of elements in the hash table and add them to an array.

For each word in the list:

We sort the character of the words:

Use this sorted string to find out where we insert into;

for each key in the table:

for each word in the list for the key:

add word to array;

Implementation:

sortChars(s);

if (!hash.containsKey(key)) {

hash.put(key, new LinkedList<String>());

linkedList<String> anagrams = hash.get(key);

anagrams.push(s);

Code:

Test cases:

Analysis: O(n)

**Problem 4:** Search for string in an array interspersed with empty. –done

Assumptions: can search for empty string.

Idea: Modification to binary search: in case mid is an empty string, move to first non-empty string

**Problem 5:** Binary search: N by M matrix. Array is sorted. Method to find elt. Approach 1: bring observations together here:

If the start of a column is greater than x, then x is to the left of the column.

If the end of a column in less than x, then x is to the right of the column

If the start of a row is greater than x, then x is above that row.

If the end of a row is less than x, then x is below that row.

We now have a sub-matrix to consider that looks like the following (the gray squares have been eliminated).

We can repeatedly apply these conditions to search for S5. Note that the only conditions we actually use are conditions 1 and 4.

Approach 2: Look along the diagonal and do a binary search. Split 2D array into top left and bottom right. Recurse on top right and bottom left.

**Problem 6:**

N integers rotated: ordered normally.

{10, 15, 20, 0, 5}

{50, 5, 20, 30, 40}

Both arrays have midpoint of 20, but 5 appears on the left side of one and right side of other. Comparing x with the midpoint is inefficient.

One part of the array must be ordered normally. Look at the normally ordered half to determine whether to search the left half or the right half.

If left < middle: left is *ordered normally*.

If left > middle: right is *ordered normally*.

if(x == a[mid]) return mid;

if(a[left] < a[mid])

{

if( (x >= a[left] && a[mid])

return search(a, left, mid -1, x);

if ( right < left) return -1;

return search(a, mid+1, right, x);

} else if (a[mid] < a[left] )

{

if (x >= a[mid] && x <= a[right])

return search(a, mid+1, right, x);

}

hello world

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

***Problem 8:*** Find magic index A[i] .

Binary search, except the below:

if (A[mid] < mid ) search right;

else search left;

Must be on right, because value must decrease by 1.

With dups:

Idea:

Algorithm:

min(midIndex - 1, midValue); 🡪 search left

max(midIndex +1, midValue); 🡪 search right

leftIndex = min(mid-1, midVal)

left = magicFast(array, start, leftIndex)

if(left >=0) return left;

rightIndex = max(midIndex +1, midValue) ;

right = magicFast(arr, rightIndex, end);

return right;

magic index :

2, 1, 3 return 1;

-1, 1, 4, 5, 6

Idea: if A[mid] < mid , then magic index must be to the left of index mid in the array .

int findMagicIndex(int a[], int n)

{

int i;

for(i = 0 ; i < n; i++)

if(A[i] = i) return i;

return -1;

}

int magicFast(int a[], int lo, int hi)

{

if ( lo < hi || lo < 0 || hi >= n) return -1;

int mid = (lo + hi) /2;

if(a[mid] == mid)

return mid;

else if (a[mid] > mid)

return magicFast(a, lo,mid -1)

else

return magicFast(a, mid+1, hi);

}

duplicates : leftIndex: min(midIndex -1, midValue) , rightIndex : min(midIndex +1, midValue);

A[5] > 20;

40 -20 -1 1 2 3 5 7 9 12 13

***Problem 9:***

We have a list of items. Find the longest sequence such that both the first and second items are in non-decreasing order.

Apply the Simplify and Generalize approach (or Pattern Matching approach), relate this problem to find the longest increasing sequence in any array.

Longest increasing subsequence which ends with i. Can be found by looking at all the prior solutions. Append A[i] to the longest valid one, where valid means any list where A[i] > list.tail.

1. If elts do not need to stay in same order, the problem is trivial.

longest(*ending with* A[0]): 13

longest(*ending with* A[1] ): 13,14

longest(*ending with* A[2]): 10

longest(*ending with* A[3]): 10, 11

longest (*ending with* A[4]): 10, 11, 12

Sort the list of people by their heights (can use sort library function), and apply the longestIncreasingSubsequence to just their weights. (\*\* go over \*\* ).

void longestIncreasingSubsequence(ArrayList<HtWt> array, ArrayList<HtWt> solutions, int current\_index);

**Problem 10:**

Goal: do an in-order traversal, keep a counter as we traverse. By the time we find x, counter will equal the number of elements less than x.

Keep counter at each node, indicating the number of nodes in the left subtree. As long as we’re moving left, counter won’t change. Why? All elements to the left are less than than x, so we’ll need to increment counter by number of elements in the left subtree. When we move to the right though, we’ll need to increment the counter by the number of elements in the right subtree.

For counting the number of elements in the right subtree, we can track this information as we add new elements to the tree.

Finally, we compare 24 with node 25 and find that 24 must be on the right. We’d compare 24 with the root, 20, and find that 24 must reside on the right.

int getRank(Node node, int x):

if x is node.data

return node.leftSize();

If x is on left of node

return getRank(node.left, x);

If x is on right of node

return node.leftSize() + 1 + getRank(node.right, x);

**Problem 12:** Find the kth to largest element in an unsorted array.

Idea: pick a pivot randomly, divide into all elements below and above. As you divide, go into the partition where y-l + 1 (\*\* review this\*\*).

qselect(a, l, u, r):

y = part(a, l, u); // a[l .. y -1] < a[y] , a[y] < a[y+1 .. u]

if(y – l + 1 == r ) return a[y];

if(r < y – l +1)

qselect(a, l, y-1, r);

else

qselect(a, y+1, u, r – (y – l + 1));

**Problem 7:** Phone numbers

**Trees and Graphs:**

Notes from OCW

*Binary Tree vs. Binary Search Tree:* A binary search tree imposes the condition that, for all nodes, the left children are less than or equal to the current node, which is less than all the right nodes.

*Balanced vs. Unbalanced:* Balanced tree is one in which depth of subtrees vary by a certain amount. It does not mean that left and right subtrees are exactly the same size.

*Full and complete trees:* All leaves are at the bottom of the tree, and all leaves have exactly two children.

*Binary tree traversal:*

  Inorder traversal: visit left side, current, then the right (most common)

  postorder traversal: visit left, right, current

*Balanced trees:* Red-black vs. AVL tree

*Tries:*

N-ary tree where each path down the tree leads to an n-ary word.

*DFS:*

Visit a node r and then  iterate through each of r's adjacent nodes. When visiting a node n that is adjacent to r, we visit all of n's adjacent nodes before going on to r's adjacent nodes. Each node n is searched before looking at its children.

(Pre-order traversal is a form of DFS. The main difference between DFS and pre-order traversal algorithm is that in DFS we must mark nodes visited. Note that if we don't mark nodes visited in DFS, we risk getting stuck in an infinite loop.)

Use it when: Visit every node until we find what we're looking for.

*BFS:* Visit each node r's adjacent nodes before searching any of r's grandchildren. Implemented with an iterative solution using a queue.

Use it when: Want to quit searching if we get too far from the root.

/\* Given a binary tree, print its nodes in preorder\*/

void printPreorder(struct node\* node)

{

     if (node == NULL)

          return;

     /\* first print data of node \*/

     printf("%d ", node->data);

     /\* then recur on left sutree \*/

     printPreorder(node->left);

     /\* now recur on right subtree \*/

     printPreorder(node->right);

}

// An iterative process to print preorder traversal of Binary tree

void iterativePreorder(node \*root)

{

// Base Case

if (root == NULL)

return;

// Create an empty stack and push root to it

stack<node \*> nodeStack;

nodeStack.push(root);

/\* Pop all items one by one. Do following for every popped item

a) print it

b) push its right child

c) push its left child

Note that right child is pushed first so that left is processed first \*/

while (nodeStack.empty() == false)

{

// Pop the top item from stack and print it

struct node \*node = nodeStack.top();

printf ("%d ", node->data);

nodeStack.pop();

// Push right and left children of the popped node to stack

if (node->right)

nodeStack.push(node->right);

if (node->left)

nodeStack.push(node->left);

}

}

int isBalanced(node\* n)

{

int leftHeight;

int rightHeight;

// define height of empty of 1

if(abs((leftHeight = isBalanced(n->left)) - (rightHeight = isBalanced(n->right))) <= 1)

return 1 + max((n->left)->key\_value,(n->right)->key\_value);

else

return 0;

}

// Assumptions:

// Idea: Use BFS to search for a path from a node n1 to the second node n2.

// Test/correctness:

// Analysis:

bool pathbwnodes(Graph g, GraphNode n1, GraphNode n2)

{

if (n1 == n2) return true; // handle case when start and end node are the same.

LinkedList<Node> q = new LinkedList<Node>();

for (Node u : g.getNodes())

u.state = State.Unvisited;

start.state = State.Visiting;

q.add(start);

Node u;

while(!q.isEmpty()) u = q.removeFirst();

if(u != null) {

for (Node v : u.getAdjacent())

{

if(v== end)

return true;

else

v.state = State.Visiting;

q.add(v);

}

}

u.state = State.Visiting;

return false;

}

void printPostorder(struct node\* node)

{

     if (node == NULL)

        return;

     // first recur on left subtree

     printPostorder(node->left);

     // then recur on right subtree

     printPostorder(node->right);

     // now deal with the node

     printf("%d ", node->data);

}

/\* Given a binary tree, print its nodes in inorder\*/

void printInorder(struct node\* node)

{

     if (node == NULL)

          return;

     /\* first recur on left child \*/

     printInorder(node->left);

     /\* then print the data of node \*/

     printf("%d ", node->data);

     /\* now recur on right child \*/

     printInorder(node->right);

}

// Idea:

Node inorderSucc(Node n){

if (n == NULL) return NULL;

if(n.right != null) return leftMostChild(n.right);

else

{

TreeNode q = n;

TreeNode x = q.parent;

while((x != null) && (x.left !=q)

{

q = x;

x = x.parent;

}

}

return x;

}

}

public TreeNode leftMostChild(TreeNode n)

{

if( n == null) return null;

while(n.left != null) n = n.left;

return n;

}

class commonAncestProb {

public:

boolean covers(TreeNode root, TreeNode p)

{

if (root == NULL) return false;

if (root == p) return true;

return covers(root.left, p) || covers(root.right, p

}

TreeNode commonAncestorHelper(TreeNode root, TreeNode p, TreeNode q)

{

bool is\_p\_onleft = covers(root.left, p);

bool is\_q\_onright = covers(root.right, q);

// If p and q are on different sides of the root, return root as the common ancestor

if(is\_p\_on\_right != is\_q\_on\_right) return root;

TreeNode child\_side = is\_p\_on\_left ? root.left : root.right;

return commonAncestorHelper(child\_side, p, q);

}

TreeNode commonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(!covers(root, p) || !covers(root, q))

return NULL;

return commonAncestorHelper(root, p, q);

}

class graph

{

public:

typedef map<string, vertex \*> vmap;

vmap work;

void addvertex(const string&);

void addedge(const string& from, const string& to, double cost);

};

void graph::addvertex(const string &name)

{

vmap::iterator itr=work.begin();

itr=work.find(name);

if(itr==work.end())

{

vertex \*v;

v = new vertex(name);

work[name]=v;

return;

}

cout << "\nVertex already exists!";

}

void graph::addedge(const string& from, const string& to, double cost)

{

vertex \*f=(work.find(from)->second);

vertex \*t=(work.find(to)->second);

pair<int,vertex \*> edge = make\_pair(cost,t);

f->adj.push\_back(edge);

}

struct node

{

// int key\_value;

int key\_value;

node \*left;

node \*right;

bool visited;

Method to find the location of a given.

**Heaps:**

**ADT : heap . complexity**

An array visualized as a nearly complete tree

Insert (s, x);

Max(S): return element of S with the largest key.

extract\_max(S): return element of S with the largest key, and then remove it from S. // returns elt with largest key .

increase\_key(s, x, k): increase the value of x’s key to the new value k.

*Heap:* An array *visualized* as a nearly complete binary tree. An implementation of a priority queue.

Array example (draw out) :

16, 14, 10, 8,7

Not a full binary tree : 16

Enumerate nodes starting from 1. Level order traversal starting index at 1.

OR Mapping of heap to tree corresponds to :

Parent = i/2 Left(i) = 2i / right(i) = 2i + 1 . first element I = 1.

Symmetric with min-heap.

Max-heap property : The key of a node x in the tree is greater than or equal to the key of the node x’s

children.

Example:

Max : trivially performed . want to maintain rep-invariant.

Extract-max : not trivially performed .

Build a max-heap out of an initially unsorted array.

Need to consider arrays that are input to sorting algorithm that aren’t heaps.

Heap operations : build\_max\_heap: produces a max\_heap from an unordered array.

Max-heap property : key of a node is >= key of children

Value of root greater than children.

Max\_heapify: (sounds strange, but ify anything) : correct single violation of heap property in a subtree’s root.

Contract ?

Assume heap means max-heap. Precondition for Max\_heapify(A, i) routine: Assume that the trees rooted at left(i) and right(i) are max\_heap. If precondition is true, you have to return a heap which has the violation x corrected.

Do recursively to build a max-heap.

Single violation , fix.

Max\_heapify(A, 2);

heap\_size(A) = 10; // ?

Exchange A[2] with A[4] .

Call max\_heapify(A, 4);

Exchange A[4] with A[8]

(No more calls.)

TODO: write code.

Complexity of max-heapify: (know how to explain: not an unbalanced tree

) 🡪 height of visualization tree is bounded by lgn + need to consider the pre-condition, i.e., assume that left[i] and right[i] are max\_heaps. .. O(lgn) 🡪

Problem: Convert an array A[1….n ] into a max-heap.

Build\_max\_heap(A):

For I = n/2 downto 1 : //

call/do max\_heapify(A, i); // everytime you call it, you satisfy the precondition.

Question: why n/2? Because the children are 2i and 2i + 1 .

Elements A[n/2+1 … n ] are leaves of heap.

Why downto? A[5] : one operation –small tree that’s a max-heap . everytime you call it, you satisfy

The precondition. Do a bunch of other things that all work on leaves. Only working with max heaps as your left child and right child.

Working your way up. Only working with max-heaps as your left child and right child.

Simple analysis: Increase in operations as you go higher up. Observe max\_heapify takes O(1) for nodes that are 1 level above the leaves, and in general O(L) time for nodes that are L levels above leaves.

Given the same algorithm: The analysis upper-bound. Can you do a better analysis that gives a better complexity?

Observations: Increase in operations as you go higher up, but fewer nodes as you go higher up.

1. Observe max\_heapify takes O(1) for nodes that are one level above the leaves and in general O(l) time for nodes that

are l levels above.

2. n/4 nodes at 1 level above the leaf, n/8 two levels above the leaf

n/4 (1 c) + n/8(2c) + n/16(3c) + … + 1(lgn c)

To make easy to reason about, set n/4 = 2^k . 🡪 quick simplification : c\*2^k ( ½ ^ 0 + 2/2 + 3/2 + (k+1)/2k );

Now, you can say complexity is theta(n), a more tight bound.

extract\_max :

Assume that left(i) and right(i) are max\_heaps.

Heap as a tree : root of tree - first element ( I = 1)

Parent (i) = i/2 ;

Left(i) = 2i right (I ) = 2i + 1 .

Priority Queue: Implements a set

**Stacks and Queues:**

*Basics:*

Implementing a stack:

Implementing a queue :

Problem: Use an array to implement 3 stacks.

Problem: Find the minimum of a stack

**Linked Lists:**

Creating a Linked List | Deleting a Node in a singly linked list.

The runner / second pointer technique

The fast node might be iterating through the list by a fixed amount.

The fast node might be hopping multiple nodes for each node the slow node iterates through.

A1: a2, a3, b1, b2, b3

A1: b1 a2 b2 a3 b3

Problem: find all duplicates of a linked list.

Problem: find the kth to last element of a linked list.

**Arrays:**

*Basics:*

Dictionary: Abstract Data Type (ADT)

Maintain set of items, each with a key.:

Do all operations in

insert(item): (overwrite any existing key)

delete(item):

search(key): return item with given key or report doesn’t exist.

*O(lg n) via AVL*

*Motivation:*

* *docdist*
* *databases : hashing vs. search tree*
* *Berkeleydb - hash database*
* *Dictionary: closest dictionary spelling Tweaking letters - trialperturbations of letters*
* *Search engines (old days)*
* *Compilers and interpreters : dictionary with all variables*
* *Network ( router needs to know all machines connected to it): subnet*
* *Searching in a string : control s - hashing to search for string . using algorithms covered.*
* *String commonalities*
* *File and directory synch.*
* *Cryptography.*

Simple approach: direct-access table

Big table: array.

Store items in array.

Indexed by key.

Badness of the above:

1. Keys may not be non-neg. integers ,e.g., a[-2]

Solution to 1: pre-hash

* 1. Maps keys to non-neg. integers
  2. In theory, keys are finite and discrete, so

1. gigantic memory hog:
   1. hash ( ‘0B’) = hash(‘ ‘);
   2. hash(x) = hash(y) ; x = y;
   3. \_\_hash( ) : don’t mess with rehash function.

Etymology of hash :

Why we use that term:

Hashing: reduce size of all keys (integers) down to reasonable size m for table.

Idea: m = O(n) - #keys in dict.

Collision : h(k\_i) = h(k\_j) ;

Chaining: if multiple hash values with the same value, store as a list.

Worst case: theta(n) - unless you’re really unlucky.

Each key is equally likely to be hashed to any slot in the table, independent of where other keys hashing.

Expected length of a chain: for n keys, m slots.

Prove this, but also going to cop out.

Hashing function: O( 1, .. , n) ;

Simple uniform hashing:

Question on low space. But, how do we get low time ? two probabilistic args

Each key is equally likely to be hashed, independent of where other keys are hashing.

Analysis: expected length of chain of n keys, m slots.

1/m + 1/m + 1/m …. 🡪 n/m : load factor of the table (note that it’s not (1/m)\*(1/m)\*(1/m) ) ;

O(1) if m = theta(n);

Insert / delete: theta(1)

Search: 1 + length of chain;

All independent = alpha;

Worst case : theta (n);

theta(1) if m = n ;

Hash functions:

1. Division method: h(k) = k mod m. good if m’ is prime.
2. Multiplication method :
   1. H(k) = a\*k mod 2^w >> ( w –r)

*Find a peak:*

*Linear search*

*Binary search*

A is a 2D peak iff a >= b , a>=c , a>= e , a>= 3,

Greedy ascent: look up as much as possible O(nm ) complexity

Divide and conquer:

Pick middle column: find 1D peak

Use i,j as a start to find a 1D peak on row i.

Pick middle column j = m/2

Find global max on column j at (i,j)

Compare (I, j-1) , (I,j), (i, j+1)

Pick left cols if (I, j-1) > (I,j) . similarly for right .

if(I,j) >= (I,j-1) , (I, j+1) 🡪 (I,j) is a 2D peak

T(n, m) = T(n, m/2) + O(n) ; 🡪 O(nlog2m) ;

solve the new problem with half the number of columns. Matches with 2D peak.

elements on left and right :

When you have a single column, find the global maximum, and you’re done.

**Problem:** Implement an algorithm to determine if a string has all unique characters. What if you cannot use additional data structures?

**Solution:**

*Assumptions:*

* Unique character means that no character appears more than once in the array.
* If the string is of size 0, then the string has all unique characters.

*Idea for part 1:*

If you can use additional data structures, you put the characters in a hash table, and then check each character for a hash collision.

*Code:*

*Idea for part 2:* If one cannot use additional data structures, you run through each element I of an array, and check whether any other element in the array has a value equal to element I’s value. If it does, then you stop searching for other duplicates and print to the user that a duplicate has been found.

*Code:*

int main(int argc, char\*\* argv)

{

// get the string as input

char\*\* inputString = (atoi) argv[1];

int seenPreviously = 0;

for (int i=0; i<n; i++)

for (int j=i+1; j<n; j++)

if ((inputString[i] == inputString[j]) && i !=j )

{

seenPreviously = 1;

printf(“The array provided has a duplicate. The duplicate is at array element %d and array element %d\n.” i, j);

return;

}

printf(“The array provided has a duplicate. The duplicate is at array element %d and array element %d\n.” i, j);

}

*Test*:

Input of null string: Input of no duplicates: Input of all the same numbers: Input with string buffer overflowing:

*Analysis:*

The first code runs in O(n) time. Each hash table operation is a constant. The second code runs in O(n2) time. The inner loop to match an element takes O(n) time and the outer loop to match an element takes O(n) time.

Related: Prefix sum

for (i = 0 ; i < n ; i++)

b[i] = b[i-1] + a[i];

Problem: Game of Life:

**Jacobi Problem:**

*Background:*

*Problem:* Simulate heat diffusion such that on each timestep, heat spreads more evenly across a 2D plane.

*Use case:* heat diffusionfor simulation of plasma-coupled combustion to generate energy for longest time.

See code documentation for application.

*Idea:*

*Code:*

*User-friendly function:*

#define DIVISOR 5.0

void jacobi(int n, int m)

{

while(err != maxerr )

{

MPI\_Isend();

MPI\_Irecv();

MPI\_Waitall();

jacobi\_sweep(a, anew);

MPI\_Allreduce();

}

}

void jacobi\_sweep(float\*\* a, float\*\* a\_new)

{

int divisor = DIVISOR;

getslack();

#pragma omp parallel

FOR\_ALLBEGIN()

for (int j = 0; j< m; j++)

for (int i = 0 ; i < n ; i++)

a\_new[i][j] = a[i] + a[i-1][j] + a[i+1][j] + a[i][j+1] + a[i][j-1]/DIVISOR;

for\_ALLEND()

getslack();

#pragma omp parallel

{

FOR\_ALLBEGIN() --- expand below

for (int j = 0; j < m; j++)

for (int i = 0; i < n; i++)

a[i][j] = a\_new[i][j];

for\_ALLEND();

}

endLoop();

}

Notes on Architecture:

GPU hardware:

GPU direct:

GPU 3D memory:

int main(int argc, char\*\* argv)

{

if(argc > 5)

{

n = atoi(argv[1]);

m = atoi(argv[2]);

}

a[i][j] = fread(matrixiputFile);

MPI\_Init(&argc, &argv);

MPI\_Comm\_Rank();

MPI\_Comm\_size();

PAPI\_start(PAPI\_L2\_DCM);

double startTime = get\_wtime();

// Boundary

answer = jacobi(n, m);

double endTime = get\_wtime();

PAPI\_end(PAPI\_L2\_DCM);

// Per-function profiles

GetPAPIHWctrs();

writeResToFile(endTime -startTime,

enddcm – startdcm,(answer – actualAnswer)< MAX\_ERR?true:false));

WriteProfileResultsToFile();

MPI\_Finalize();

}

OpenACC version:

C:

Fortran:

#pragma acc parallel pcopyin( a\_new[1:n]) pcopyout(a[1:n], halo[1:n]) - forced

#pragma acc kernels - advisory

const :

allocatable vs. pointer:

Same as five-finger rule for OpenMP

MPI+OpenMP+OpenACC version:

Can adjust CPU fraction / GPU fraction to schedule on CPU and GPU simultaneously.

*Complexity:* 0(n\*m);

*TestCase:* 16 by 16, 32 by 32, follow method of testing that was done.

*Compile and link:*

*Documentation for developers:*

*Git repo:*

*Plots:*

*Paper:*

*Related:*

*Tm:*

*Intro:*

All cells of the 2D matrix are initialized at 50.0. The mesh computation is simulating heat dissipation from one endpoint to another on a 2D grid. The top left and bottom right corners of the matrix have values of 0.0 and 100.0, respectively. On each timestep, the right two endpoints get reset. Also, the boundary condition is set to 0.0. With this, I check the correctness of the answer through printing a sample of the matrix, specifically a cell in the top left region and a cell in the bottom right region.

One note is that the profiling was incorrect. This doesn’t change the timings luckily.

Golden for 128 is [ 45.412, 53.722 ].

Golden for 512 is [ 45.412, 53.722  ].

I run the mesh computation program with problem size of 128 by 128 for 10,000 timesteps, using x=16 repetitions. The experiment was run on 1 node of Surface using 1 GPU with 2880 GPU cores each, 1 MPI process per node.

**Using -g/-O0:** with scheduling of gpu threads

Non-functionized version with allocatable:     11.202 seconds        CUDA memcpy HtoD + DtoH:  0.365 secs (5.01%)           sweep:   4.625 secs (61.50%)                     sweep copy: 2.142 secs (28.49%)           [45.463, 53.722]

Non-functionized version with pointer:                  - seconds           CUDA memcpy HtoD + DtoH:  -       (5.06%)           sweep:   20.092 secs  (61.66%)                     sweep copy:  16.695 secs (28.46%)     [45.461, 53.722]

Functionized version with allocatable:           14.833 seconds          CUDA memcpy HtoD + DtoH: 0.366 secs   (4.96%)         sweep:  4.664 secs (61.73%)                     sweep copy: 2.138 secs (28.78%)           [46.019  53.276]

Functionized version with pointer:                 14.652 seconds          CUDA memcpy HtoD + DtoH: 0.368 secs   (4.87%)         sweep:  4.666 secs (61.82%)                     sweep copy: 2.137 secs (28.32%)           [46.019  53.276]

**Using -O2 : ?**

Non-functionized version with allocatable:    6.312 seconds         CUDA memcpy HtoD + DtoH: 0.241 secs (11.52 %)           sweep:   0.976 secs (42.59%)                 sweep copy: 0.869 secs (37.92 %)                [45.463  53.722]

Non-functionized version with pointer:          42.580 seconds       CUDA memcpy HtoD + DtoH: 0.391 secs (5.06 %)             sweep:   20.09 secs (61.66%)                   sweep copy: 16.99 secs (28.46%)               [45.463  53.722]

Functionized version with allocatable:          9.894 seconds          CUDA memcpy HtoD + DtoH: 0.378 secs (4.96%)             sweep:  0.957 secs (61.33%)                  sweep copy: 0.863 secs (28.78%)                  [46.019 53.276]

Functionized version with pointer:                12.444 seconds        CUDA memcpy HtoD + DtoH: 0.379 secs (1.21%)             sweep:  3.396 secs (61.58%)                  sweep copy: 1.273 secs (24.17%)                  [46.019 53.276]

With -O3, the answer becomes different enough and the timings aren’t actually that much better, so I stop at -O2. Note that the time remaining is kernel launch overhead, i.e., CuKernelLaunch.

I run the mesh computation program with problem size of 512 by 512 for 10,000 timestep, using x=16 repetitions. The experiment was run on 1 node of Surface using 1 GPU with 2880 GPU cores each, 1 MPI process per node.

**Using -g / -O0:**

Non-functionized version with allocatable:  82.793 seconds CUDA memcpy HtoD + DtoH: 0.414 secs (0.51%)             sweep: 56.424 secs (71.41%)              sweep copy: 21.764 secs (27.54%)               [47.8586 52.2302]

Non-functionized version with pointer:  82.334 seconds CUDA memcpy HtoD + DtoH: 0.434 secs (0.52%)             sweep: 56.91 secs (71.45%)                sweep copy: 21.69 secs (27.51%)               [47.8586 52.2302]

Functionized version with allocatable:             87.382  seconds  CUDA memcpy HtoD + DtoH: 0.44 secs (0.50 %)              sweep: 56.519 secs (71.33%)            sweep copy: 21.891 secs (27.63%)                      [48.136, 51.965]

Functionized version with pointer:  87.642 seconds CUDA memcpy HtoD + DtoH: 0.46 secs (0.52 %)              sweep: 56.443 secs (71.38%)            sweep copy: 21.802 secs (27.57%)                      [48.136, 51.965]

**Using -O2:**

Non-functionized version with allocatable: 23.147 seconds       CUDA memcpy HtoD + DtoH: 0.258 secs (1.32%)           sweep:   10.985 secs (56.24%)               sweep copy: 8.043 secs (41.18%)                    [47.8565 52.230]

Non-functionized version with pointer:    -        seconds       CUDA memcpy HtoD + DtoH:  -  secs (5.06%)                 sweep:    -          secs (61.66%)              sweep copy: 3.19 secs (28.46%)                   [47.8565 52.230]

Functionized version with allocatable:         24.888 seconds         CUDA memcpy HtoD + DtoH:  0.506 secs (2.32%)           sweep:  8.894 secs   (50.65%)             sweep copy: 8.027 secs   (45.71%)                  [48.135  51.965]

Functionized version with pointer:                   65.995 seconds         CUDA memcpy HtoD + DtoH:  0.425 secs (5.01%)           sweep:  44.827 secs (61.58%)             sweep copy: 13.091 secs (28.44%)                  [48.135  51.965]

Here as well, the answer becomes different enough with -O3 and the performance isn't that much better, so I stop at -O2.

The observation from the above is that the non-functionized version performs noticeably (though still by a constant factor) better than the functionized version. The functionized version compiled with allocatable is 87.382 seconds - 82.449 = 4.933 seconds slower than that of the non-functionized version. The sweep part of the program accounts for a large portion of this difference: 62.33 seconds - 58.87 = 3.66 seconds. To get the overhead per timestep in the functionized program, we divide the difference in execution times between the functionized and non-functionized version by the number of timesteps: 4.933 seconds / 10000 = 493 microseconds of overhead per timestep. A function call isn’t more than 10 nanoseconds long, so the overhead per timestep must be coming from elsewhere, i.e., the compiler’s inefficiencies in optimizing the computation in the program. These differences suggest that the compiler is performing optimizations for the non-functionized version while giving up on optimizations in the functionized version.

A conclusion from this analysis is that function inlining in an application code makes a significant performance difference after the parallelization is taken care of for small problem A recommendation is to develop a technique to do function inlining for an application. Based on the timings above for allocatable vs. pointer, one could also consider attributes such as contiguous when declaring data arrays, though the impact for this example is small.

When using the highest level of optimization -O2, the performance changes moderately for the 128 by 128 problem and significantly by a factor for 512 by 512 problem. The relative performance (with the exception of Non-functionized with pointer) remains the same.

Questions about research:

Functionality of library:

The paper that discusses the results of a Hybrid Static/dynamic Scheduler applied to Communication-Avoiding LU: [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.400.4446](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.400.4446" \t "_blank). Applying our scheduling strategy to a CALU pthread implementation provided 34% performance gains. (In this particular paper, we also show performance improvements over two other widely used implementations of  LU factorization).   
  
The general goal underlying the low-overhead scheduling strategies in my thesis is to efficiently mitigate within-node imbalance during a scientific application's execution on current and future clusters of multi-core nodes, in order to improve application performance. More specifically, our objective is to maximize within-node load balance across cores while minimizing data movement across cores during application execution.

The core idea behind our low-overhead scheduling strategies is that each thread first executes some number of pre-assigned loop iterations of an OpenMP loop of an MPI+OpenMP application (these are the statically executed loop iterations), and then continues to retrieve and execute the remaining loop iterations from a work queue, which is shared among threads (these are the dynamically executed loop iterations). We refer to the total number of pre-assigned loop iterations across all threads (or, cores) as the 'static fraction'. We refer to the remaining number of dynamically allocated loop iterations as the 'dynamic fraction'. The dynamic fraction can be determined by a compiler, as I was mentioning. The dynamic fraction can also actually be determined by a runtime system that can adjust the dynamic fraction during application execution. Whatever method is used to determine the dynamic fraction for a particular OpenMP loop, having the ability to tune the dynamic fraction enables us to develop scheduling strategies for MPI+OpenMP applications that maximize load balance while minimizing data movement. This strategy just described is called hybrid static/dynamic scheduling (that's the one used in the above paper), and is one example of the low-overhead scheduling strategies in my thesis.   
  
An application programmer can choose from two separate methods in order to use the scheduling strategies available within our scheduling runtime system. One method is to use our scheduler functions that have some shared memory parallel programming library, i.e., pthreads, integrated into them. This is the method I was focusing on when explaining the functionality of my runtime system yesterday. The other method is to use the functions that have no shared memory library integrated. In this method, all pthreads or OpenMP threads (or generally, threads of some shared memory library) have been spawned already, and these scheduling library functions serve only to allocate loop iterations to threads.  
  
Both methods have advantages and disadvantages for different applications we have experimented with, but I will focus below on the one that does not have a shared memory programming library integrated into it, because that one helps explains the scheduler runtime system and the scheduling technique better than the other.    
  
Considering this second method of using our library, the ROSE compiler transforms the below OpenMP loop in an MPI+OpenMP application program:  
  
for (int i=0; i<n; i++)  
         c[i] += a[i]\*b[i];

so that it looks like this:

int start, end;  
#pragma omp parallel  
{  
   int myTid = omp\_get\_thread\_num();  
   int numThreads = omp\_get\_num\_threads();  
   FORALL\_BEGIN(sds, myTid, numThreads, 0, n, start, end)  
      for(int i=start; i<end; i++)  
         c[i] += a[i]\*b[i];  
   FORALL\_END(sds, myTid, start, end)  
}  
  
We use the ROSE compiler to do the source-to-source transformation of the OpenMP loop above. In this second code snippet, the ROSE compiler has inserted the FORALL\_BEGIN() and FORALL\_END() functions, which are macro functions we have defined. The ROSE compiler inserts the macro functions at the top of the application code file that contains this OpenMP loop. The macro functions make the function calls to our scheduling runtime system. The first parameter of the loop macro function identifies the scheduling strategy to be used, and its parameter value of 'sds', used in the FORALL\_BEGIN() and FORALL\_END() functions above, identify the hybrid static/dynamic scheduling strategy described a couple paragraphs earlier. Note that the lower and upper loop bounds of the OpenMP loop are changed from <0, n> to <start, end>.   
  
The second code snippet closely resembles how the OpenMP runtime system's schedulers are structured, and my runtime system could be considered a re-implementation of the OpenMP runtime system, as I was saying on the phone. However, my runtime system should not be considered a replacement for the OpenMP runtime system, due to my runtime system's limited functionality.

Problem: Implement an algorithm that reverses a string of null-terminated characters.

Solution:Reasoning: Code: Test:Analysis:

Problem: Write a method to replace all spaces in a string with %20 -.

Problem: Implement a method to perform basic string compression using the counts of repeated characters. For example, aabccccaaa would become a2b1c5a3.

Problem: Given an image represented by an N-by-N matrix, where each pixel is 4 bytes, write a method to rotate the image by 90 degrees. Can you do this in place?

Solution:

Reasoning:

Code:

Test:

Analysis:

Problem: write an algorithm such that if an element in an N-by-M matrix is 0, it’s entire row and column turn to 0.

Problem: Write a function to count the number of 0’s.

**Bit manipulation:**

*Examples with bit manip:*

*Tricks in column 3:*

1. 0110 + 0110 is equivalent to 0110 \*2 , which is equivalent to shifting 0110 left by 1.
2. 0100 equals 4 and multiplying by 4 is just left shifting by 2. So, we shift 0011 left by 2 to get 1100.
3. XOR a bit with negated value, you get 1. Axor~a = 1111
4. ~0 is a sequence of 1s so ~0 << 2 is 1s then tow 0s.

If you didn’t see these tricks immediately, think about them logically.

*Bit facts and tricks:*

Think deeply about why each is true. To understand, recall these ops occur bit-by-bity

Two’s complement and negative numbers:

Computers typically store integers in two’s complement rep.

A positive num is rep. as itself while a negative is rep.’d as 2’s comp.

*2’s complement :*

*Arithmetic vs. Logical Right Shift*

*Common Bit Tasks: getting and setting:*

*Getting a Bit of an number, i.e., integer:*

*bool getBit(int num, int i){return ((num & (1 << i )) != 0);}*

*Setting a Bit of a number, i.e., integer:*

// returns the resulting decimal number from shifting the bits

int setBit(int num, int i) { return num | (1 << i); //shifts 1 by I bits and Ors the resulting number. }

**P1: extract bits**

**P2: Binary to string: -- Convert a binary number to a string**

**Problem 3: Flip to win**

**Problem4: First:** Given a positive integer, print the next smallest and next largest number that have the same number of 1 bits in their binary representation.

**Hints:**

**Solution:**

Brute force:

Count the number of 1s in n, count the number of 1s.

Increment n until you find a number n’ having the same number of 1 bits in their binary rep.

Bit manipulation approach to get the next number .

Observation: given a number n and two bit locations I and j, suppose we flip bit I from 1 to a 0. If I > j, then n will have decreased. If I < j, then n will have increased. .

**Review/Conclusions of bit manipulation**

Slope of a line

Logic:

Circle eighth: -

Rectangle overlap:

Big-endian / little-endian:

Number of Ones:

Summary:

**Object-oriented Design ---**

Passing by Reference: modify the value passed to it.

Passing by Value: function call stack

Passing by const Reference: can’t modify the value passed to function - know what happens if not.

Design Patterns:

Singleton Method:

Factory method:

**C and C++ : virtual functions, vtable ---**

*Classes, inheritance:*

class Person {

int id;

char name[NAME\_SIZE];

}

class Student : public Person {

public:

void aboutMe() { cout << “I am a student.”; }

};

int main(){Student\* p = new Student();

p->aboutMe(); }

*Constructors and Destructors:*

Constructor of a class automatically called upon an object’s creation. If no constructor defined, compiler generates one automatically called the Default Constructor. We can also define our own constructor.

Person(int a) { id = a; }

If you just need to initialize primitive types: OR Person (int a) : id(a) { … };

*Destructor:*

Automatically called when an object is destroyed. It cannot take an argument as we don’t explicitly call a destructor.

~Person () { delete obj; // free any memory allocated within class(question: what is obj }

*Virtual function:*

In an earlie

Defined p to be of type Student:

Student \* p = new Student() ;

p->aboutMe();

What happens if we defined p to be a Person \* , like the below:

Person\* p = new Student();

p->aboutMe();

“I am a person” printed instead. Function aboutMe is resolved at compile-time, mechanism is known as static binding.

IMPORTANT: If we want to ensure that the Student’s implementation of aboutMe is called, define aboutMe in the Person class to be virtual.

class Person {

…

virtual void aboutMe(){ cout << I am a person.”;

};

class Student : public Person { public: void aboutMe() { cout << “I am a student.”; }};

Another usage for virtual functions is when we can’t implement a method for the parent class. Can Implement a common method addCourse for Student and Teacher. Calling addCourse doesn’t make sense for a person.

class person {

Int id;

char name[NAME\_SIZE];

public:

virtual void aboutMe() { cout << “I am a person” << endl; }

virtual bool addCourse(string s) = 0;

}

pure virtual function:

*Virtual destructor:*

Virtual function introduces concept of virtual destructor. Suppose we want to implement a destructor for Person and Student. A naïve solution might be ..

Since p is a Person, destructor for Person is called. Problematic because the memory Student may not be cleaned up.

Will only print:

Deleting a student.

Define the destructor for Person to be virtual.

class Person { public : virtual ~Person() { cout << “deleting a person” . ; }

class Student : public Person { ~Student() { cout << “deleting student”;}

int main() {

Person\* p = new Student();

delete p;

}

The above will output

Deleting a student.

Deleting a person.

*Pointers and References:*

A pointer holds the address of a variable and can be used to perform any operation that could be directly done on the variable such as accessing and modifying it. Two pointers can equal each other, such that changing one’s value also changes the other’s value (since they, in fact, point to the same address).

int\* p = new int;

\*p = 7;

int \* q = p;

\*p = 8;

cout << \*q;

References:

A reference is another name for a pre-existing object and it does not have memory of its own. e.g.,

int a = 5;

int& b = a;

b = 7;

cout << a;

B is a reference to a; modifying b also modifies a.

Can’t create a reference w/out specifying where in mem it referes to . However, we can create a free-standing reference:

/\* allocates memory to store 12 and makes b a reference to this piece of mem. \*/

const & int b = 12;

Unlike pointers, references cannot be null and cannot be reassigned to another piece of memory.

Pointer Arithmetic : perform addition on a pointer of struct t means skip ahead sizeof(struct t) bytes.

int \* p = new int[2];

p[0] = 0;

p[1] = 1;

p++;

cout << \*p; // outputs 1

Performing p++ will skip ahead by sizeof(int) bytes, s.t. that the code outputs 1. Had p been of diff. type, it skips ahead as many bytes as size of data structure.

*Templates*:

Templates are a way of reusing code to apply the same class to different data types.

template < class T>class ShiftedList {

T\* array; int offset, size;

Public:

Shiftedist(int sz): offset(0), size(sz) { array = new T[size];}

}

…

TODO: understand the rest of the code from book :

…

Abstract class and we can’t instantiate it.

Shallow vs. Deep Copy:

Implement a Hash table:

Virtual functions:

Smart Pointer:

Volatile:

Virtual Base Class

AlignedMalloc and Free:

2D alloc:

**Testing**

Problem: Write a function which returns hit or miss for Mastermind.

Basic parallelism:

**Threads and Locks:**

Thread and Processes: resources that they need, how it’s initiated by underlying operating system and underlying hardware.

Threads vs Processes: concurrency issues.

Threads and Processes: locks and mutexes and semaphores and monitors, and modern concurrency constructs: send/recv/waitall, collectives.

Java threads:

*Deadlock:* a thread is waiting for an object lock that another thread holds, and this second thread is waiting for an object lock that the first thread holds. Since each thread is waiting for the other thread to relinquish a lock, they both remain waiting forever.

*Deadlock detection:*

In order for a deadlock to occur, all four of the following conditions must be met

1. Mutual exclusion: Only one process can access a resource at a given time
2. Hold and wait: processes already holding a resource can request additional resources without relinquishing other resources.
3. No pre-emption: One process canoot forcibly remove another process’ resource.
4. Circular wait: Two or more processes form a circular chain where each process is waiting on another resource in the chain.

deadlock prevention:

Deadlock prevention among threads entails removing any one of the above conditions, i.e., negate one of the conditions. It gets tricky because many of these negated conditions are difficult to satisfy in the real world, e.g., many resources can be used only one process at a time, e.g., printers. Most deadlock prevention algorithms focus on avoiding condition #4, avoiding circular wait (cycle detection).

What’s the difference between a thread and a process?

How would you measure the time spent in a context switch?

Dining philosopher’s

Idea:

Create a simulation without deadlock prevention

a philosopher puts down his left chopstick if he’s unable to obtain the right one. We should not call putDown() on the chopsticks if we never had them in the first place.

correctness: One issue is that if all philosophers were perfectly synchronized, they could simultaneously pick up left chopstick, be unable to pick up right, and then put back down the left, only to have the process repeated again.

Idea2: label chopsticks with a number from 0 to N-1. Each philosopher attempts to pick the lower numbered chopstick first. This means that each philopher, except the last, goes for the left chopstick before right one (assuming that’s the way you labeled it). The last philosopher does this in reverse. This breaks the cycle.

A philosopher can never hold the larger chopstick w/out holding the smaller one. This prevents the ability to have a cycle, since a cycle means that a higher chopstick would “point” to a lower one./? (?)

**Scalability and Memory Limits**

End of Interview:

Discussing postion:

Questions:

Bye:

Facebook: Tilera , memcached

Google:

After the Interview:

Thank you note.