Introduction; Performance and Big-O

ITP 435 – Spring 2016
Week 1, Lecture 1

Lecturer: Sanjay Madhav

USC Viterbi
School of Engineering

University of Southern California

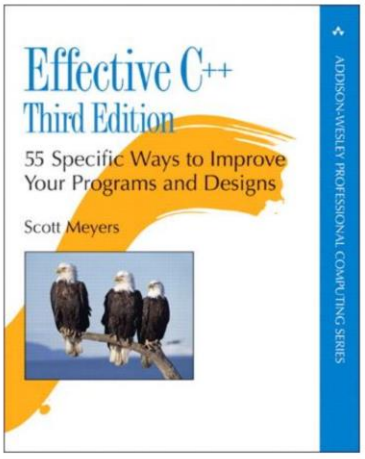
About the Instructor

- Sanjay Madhav (madhav@usc.edu)
- USC C/O 2004, BS in Computer Science, MS in CS soon, maybe a PhD in CS in the future
- Worked for 5 years in the video game industry – lots of C++
- Teaching @ USC since 2008, full-time since 2012
- I am really interested in low-level software, close to the hardware

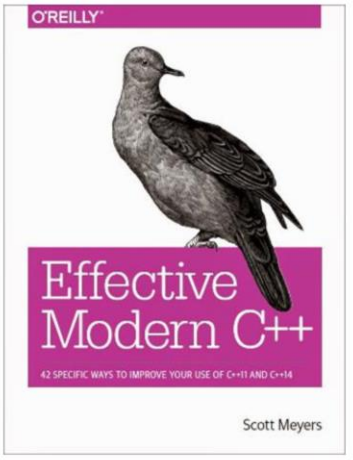
Logistics

- **Lecture:** T/Th 2:00-3:20PM in KAP 160
- Slides posted on Blackboard (but you should still come to class…)
- No discussion or lab section – ask questions during lecture!
- **My Office Hours (in OHE 530H):**
  - Monday/Wednesday 1:15-4:15PM
- **TA Office Hours:**
  - TBD

Books



Required



Recommended

Piazza

- Everyone should've received an email invite to the Piazza for this course
- Online message board
- ***All*** questions about assignments, course material, exams, etc. should go there
- ***Only*** send an email if:
  - Grade dispute
  - Personal question (such as DSP, etc.)

Piazza, Cont'd

- Try to generalize questions so that they can be ***public*** (visible to everyone)
- ...***BUT*** don't put more than 5 lines of code in a public post

- If you need more code than that for context, use a ***private*** post
- ...***BUT*** "here's lots of code, fix it!" cries for help will be ignored – you need to demonstrate that you've actually tried to debug it yourself

Grading Rubric

Category	Grade %
Programming Assignments (7% Each)	49%
Midterm Exam	25%
Final Exam	26%
Total	100%

Programming Assignments

- 7 assignments, ~2 weeks per
- Instructions on Blackboard in the "Assignments" section

- Expect each PA to take 8-12 hours – ***don't procrastinate!***

- Everything runs natively on both Windows ***and*** Mac OS X – so use whichever environment you prefer

I use automatic cheat detection software

A Couple of (True) Cheating Stories

- Student A (who took the class in a prior semester) gave their code to Student B. Student B then submitted Student A's code as their own.
- ***Result:*** Student A and B reported to SJACS for cheating and given Fs in the course
- Student C downloaded code from the Internet and submitted it as their own.
- ***Result:*** Student C reported to SJACS for cheating and given an F in the course.

USC Viterbi School of Engineering

University of Southern California

Moral of these stories is – you will get caught if you cheat. So save everyone the headache and just don't cheat.

Herb Sutter's talk on the topic: http://channel9.msdn.com/posts/C-and-Beyond-2011-Herb-Sutter-Why-C

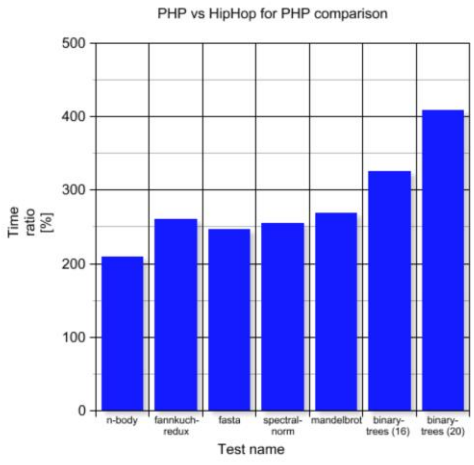
Google?

- "Most of Google is implemented in C or C++ for efficiency…" –The Anatomy of a Large-Scale Hypertextual Web Search Engine (Brin and Page, 1998)

- "Google has one of the largest monolithic C++ codebases in the world. We have thousands of engineers working on millions of lines of C++ code every day." – LLVM Blog, 2011

Facebook?

Facebook: HipHop Performance



But…why not…?

"Java is not, generally, a hard enough programming language that it can be used to discriminate between great programmers and mediocre programmers." – Joel on Software

USC Viterbi School of Engineering

University of Southern California

Full article:
http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html

But then again…

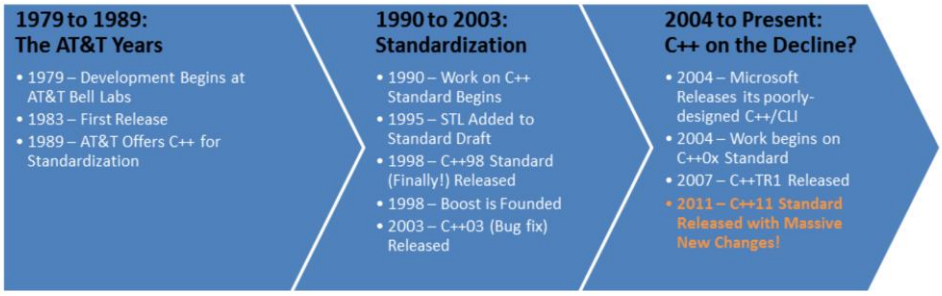
"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it, to the point where it's much much easier to generate total and utter crap with it. Quite frankly, even if the choice of C were to do *nothing* but keep the C++ programmers out, that in itself would be a huge reason to use C." – Linus Torvalds

Full rant: http://lwn.net/Articles/249460/

Goals of this Class

1. Quickly refresh your memory on the C++ you may have forgotten.
2. Refine your ability to design and write high-quality C++ code.
3. Learn new ways to use things you already know about (eg., templates).
4. Learn the ways the language is has evolved with the newer C++11 standard.

Timeline of C++ Development



On Performance and Big-O

- A fun musing based on a talk given by Bjarne Stroustrup a few years ago…you can find his talk on YouTube

- Main focus: The hardware matters!

Or...


BIG-O
the cake is a lie.
the cake is a lie.
the cake is a lie.
the cake is a lie.
the cake is a lie.

Here's a Problem

- Given ***N*** random integers, write code that inserts them into a collection in ascending order.
- Eg. if the numbers are 5, 1, 4, 2 they should appear in the collection as:
  { 1, 2, 4, 5 }
- For which ***N*** is it better to use a linked list instead of a vector?

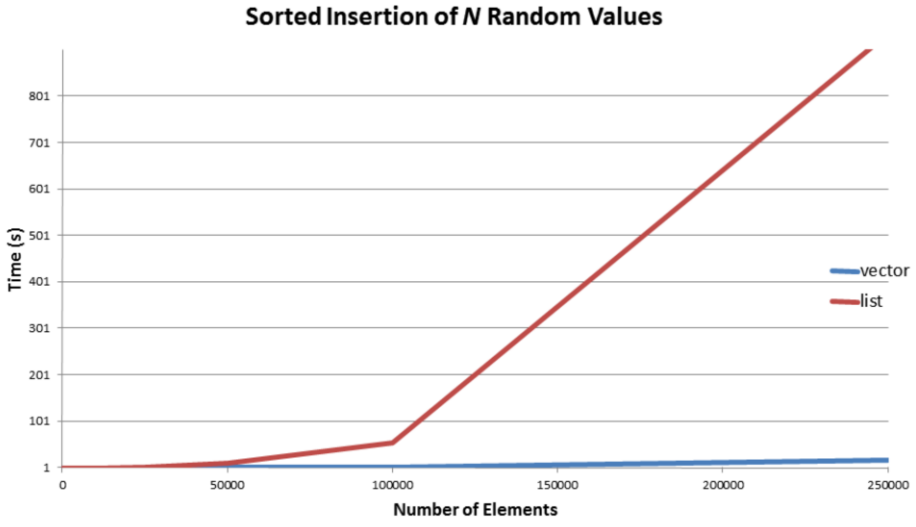
Code

- The code can be something like this:

```
Timer t;
srand(123456); // Use a set seed
std::vector<int> v;
t.start();
for (size_t i = 0; i < count; i++) {
   int number = rand() % 10000;
   auto iter = v.begin();
   auto end = v.end();
   while (iter != end) {
       if (*iter > number) {
          break;
       }
       ++iter;
   }
   v.insert(iter, number);
}
double elapsed = t.getElapsedMs();
```

Similar code for the list case

Results


Sorted Insertion of N Random Values
Time (s)
801
701
601
501
401
301
201
101
1
0
50000
100000
150000
200000
250000
Number of Elements
vector
list

But wait...

- Linear search is ***O(n)***, whether it's a vector or list

- Insertion at an arbitrary location in a list is ***O(1)***
- Insertion into an arbitrary location in a vector is ***O(n)***

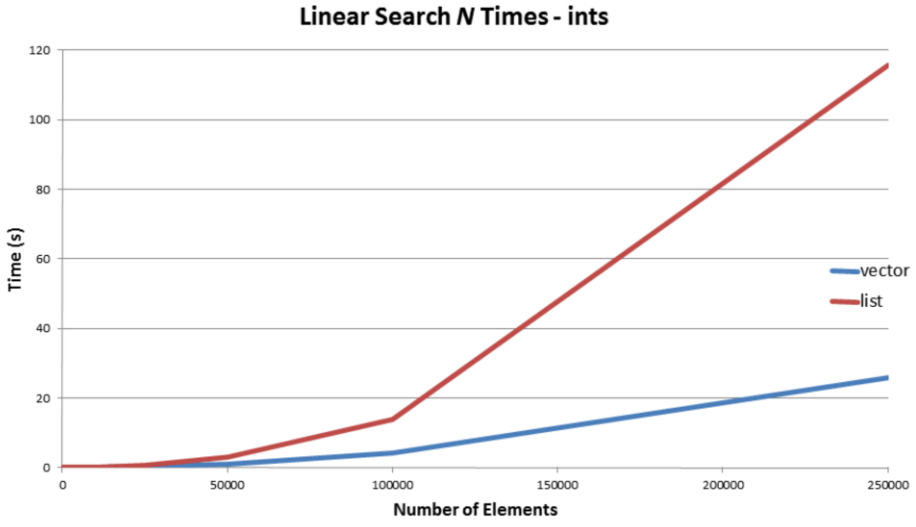
- This would lead you to believe that the list should perform better!

Linear Search of Vector/List

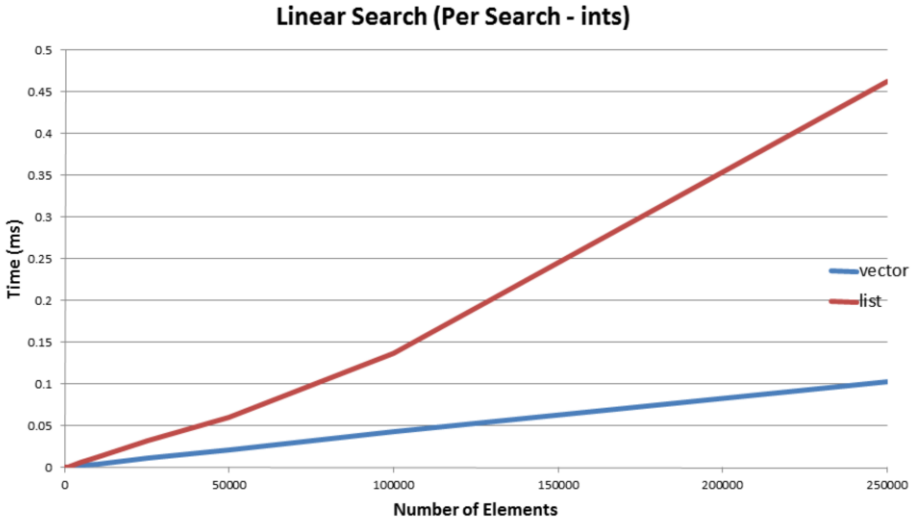
- A linear search over a vector or a list should be an ***O(n)*** operation

- However, we it turns out that all ***O(n)***s are not created equal!

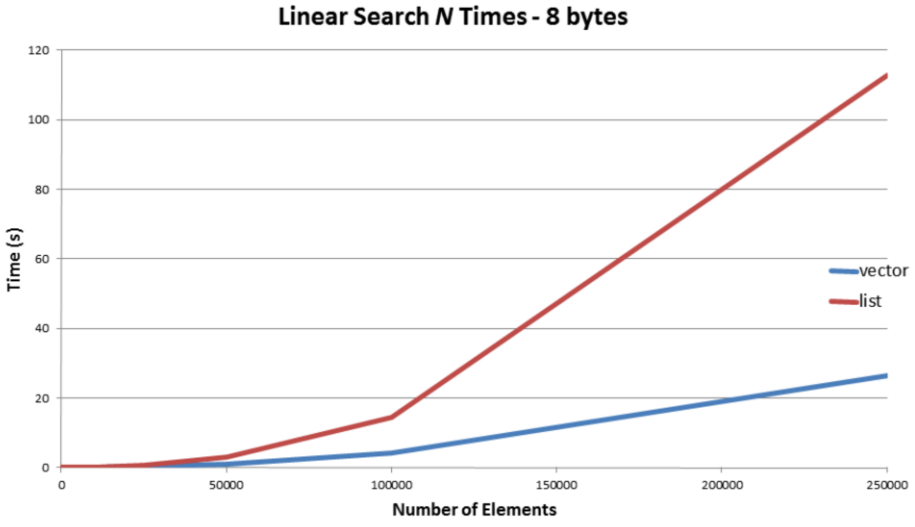
Linear Search *N* Times – ints



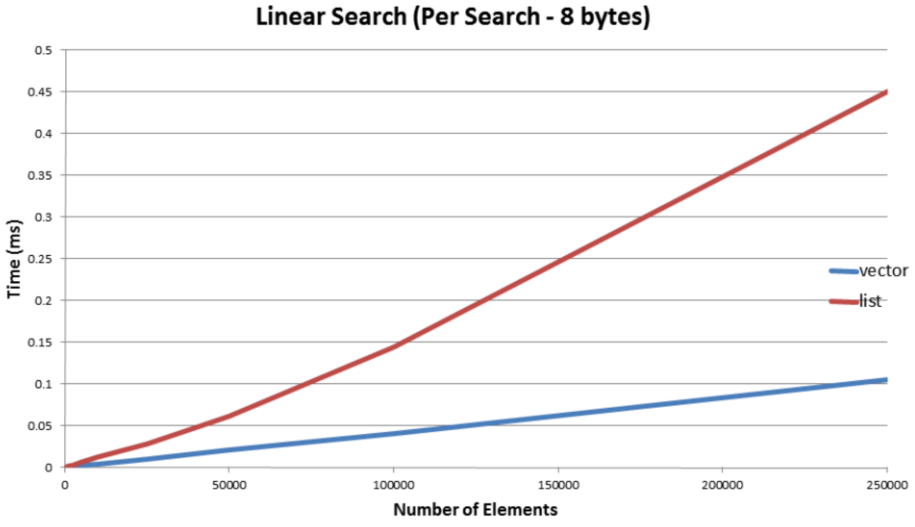
Linear Search Per Search – ints



Linear Search *N* Times – 8 bytes



Linear Search – Per Search – 8 Bytes

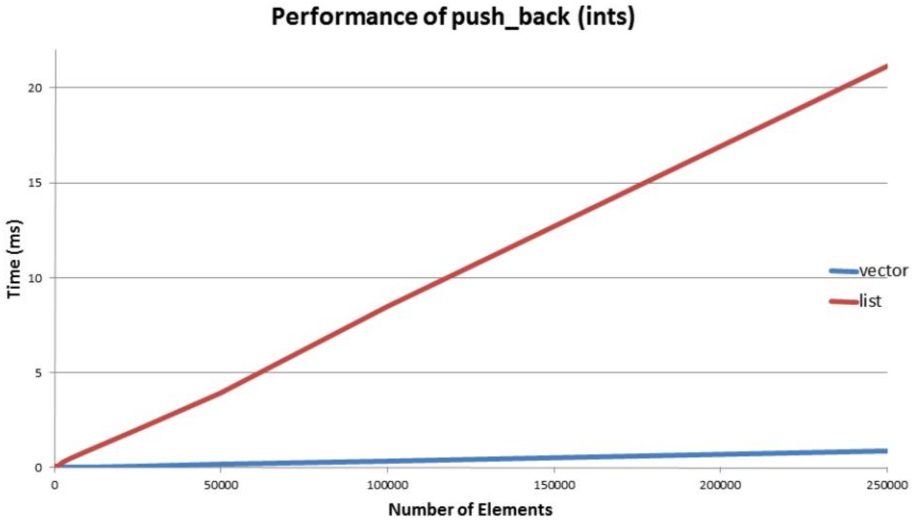


Battle of push_back

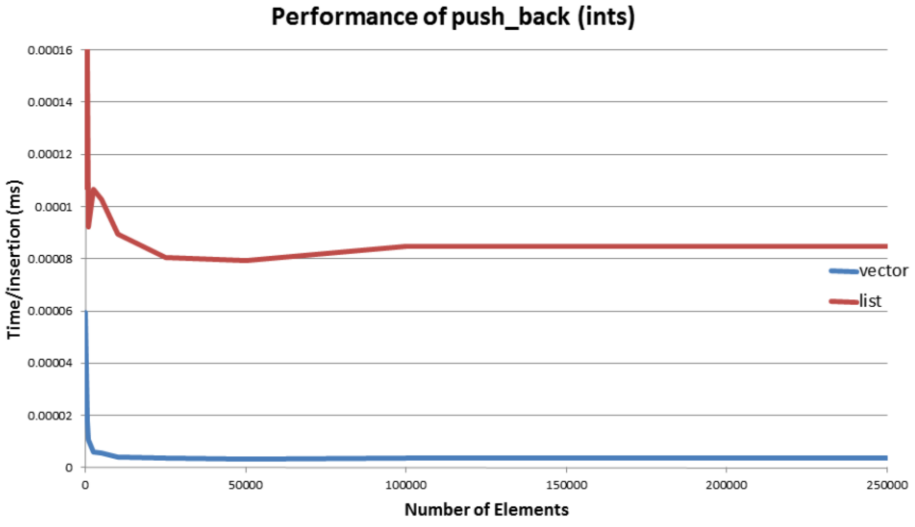
- For both a std::list and std::vector, push_back is considered ***O(1)***

- So does this mean that the performance is the same?

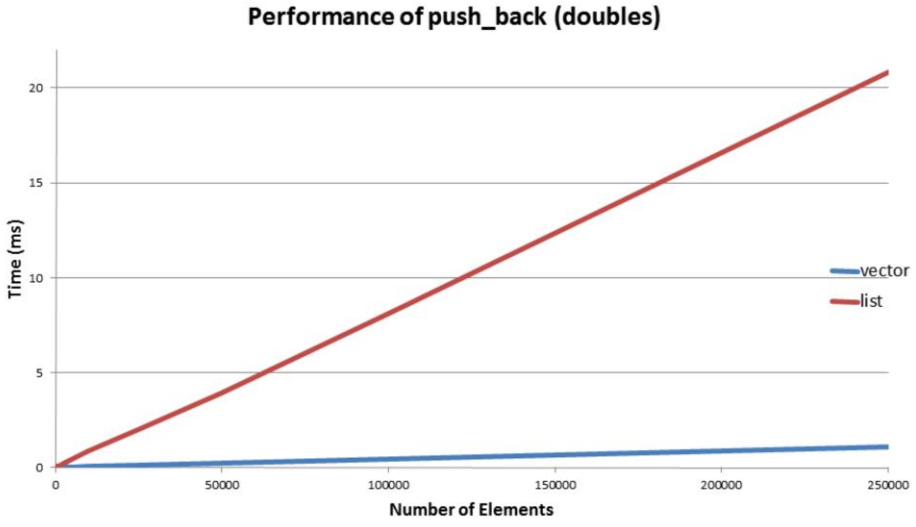
Battle of push_back, Cont'd



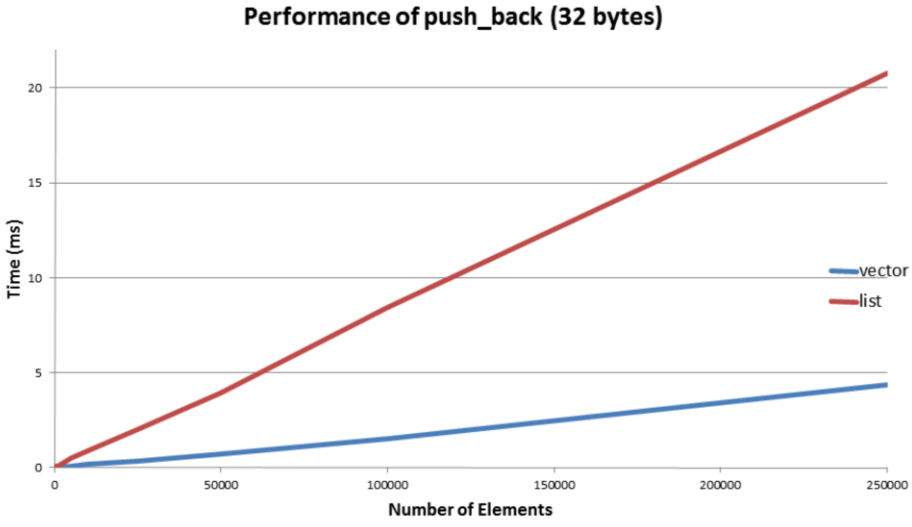
Push_back Per Element



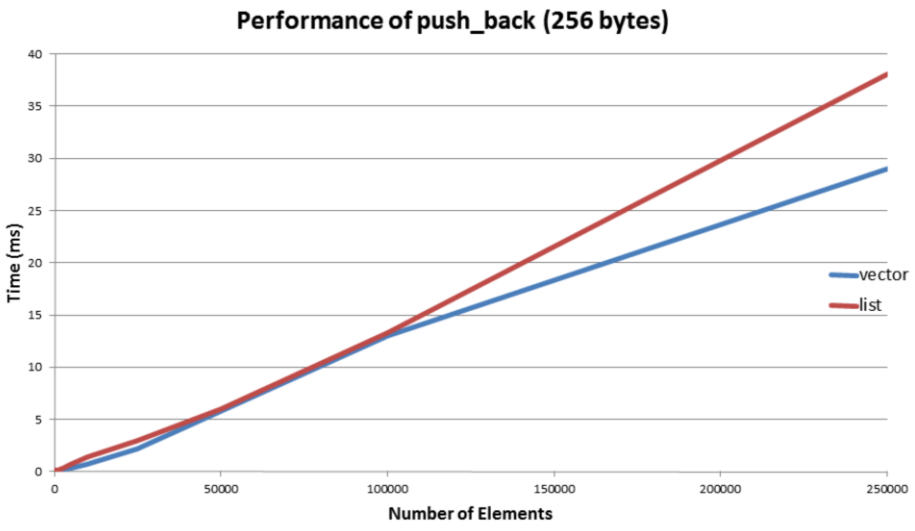
Does the Data Matter? Doubles



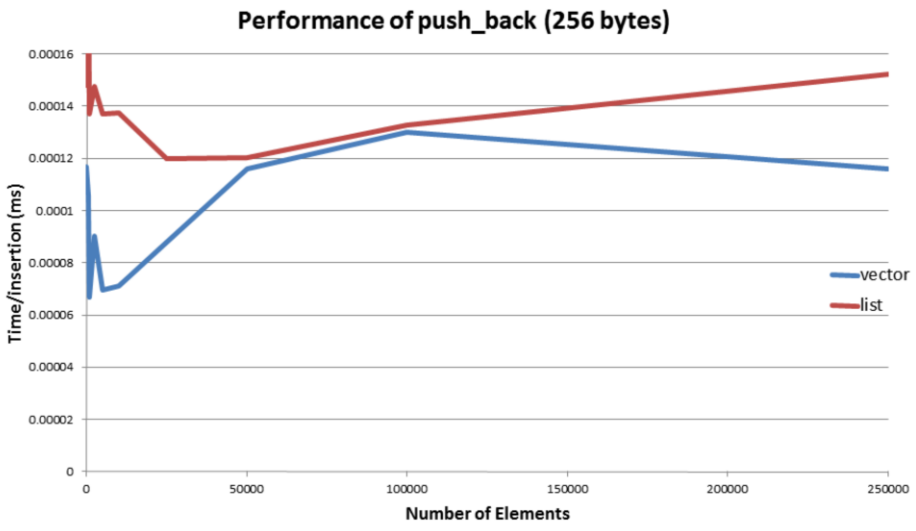
Does the Data Matter? 32 bytes



Does the Data Matter? 256 bytes



256 Bytes – Per Element



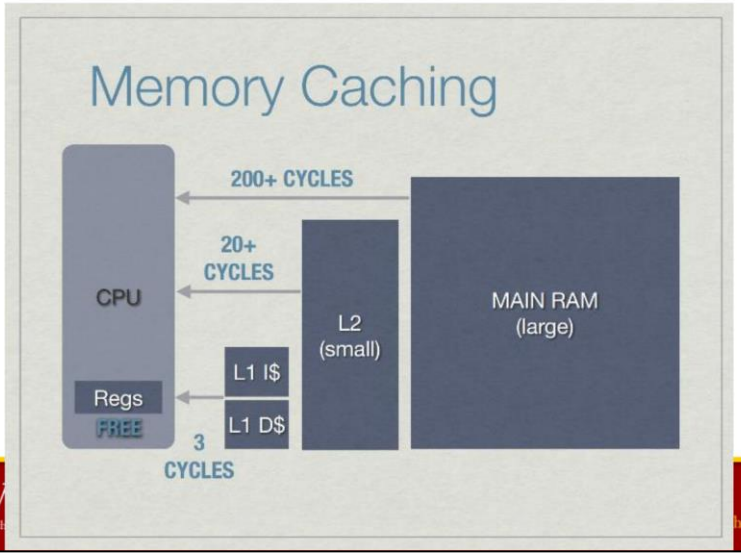
What can we infer?

Although push_back is ***O(1)*** for both list and vector...

- For data sizes 256 bytes or less, insertion into a list is more expensive
- This is likely because each push_back for list incurs a dynamic allocation, whereas vector amortizes one allocation for many elements
- The cost of copying is ***not*** greater than the cost of dynamic allocation, at least for smaller items

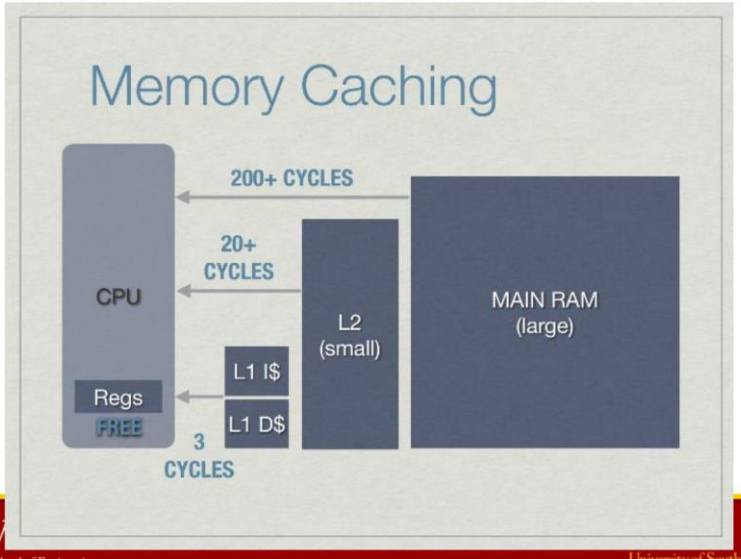
Why?

- So why are we getting these results?
- It has to do with this picture:



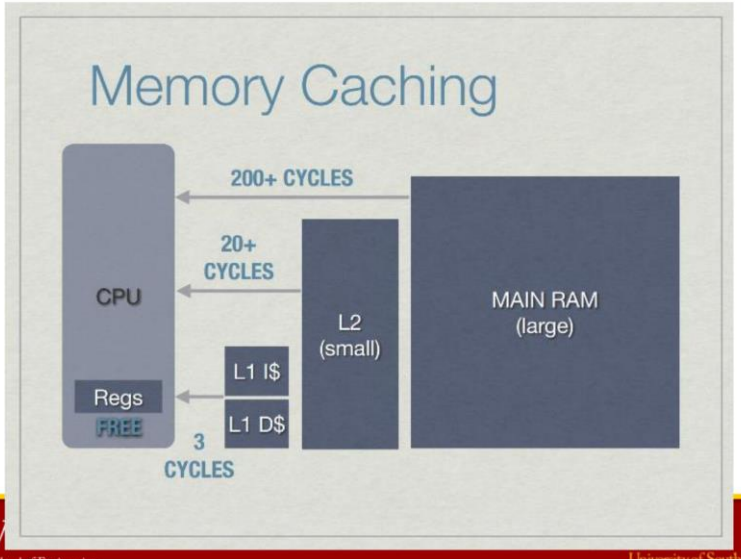
A Quick Explanation of the Cache

- If the CPU requests a value not in the cache, it's a ***cache miss***

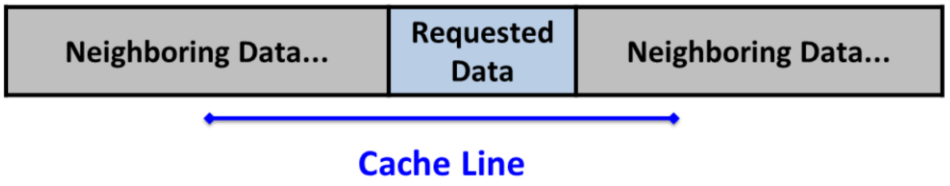


A Quick Explanation of the Cache

- If the CPU requests a value not in the cache, it's a ***cache hit***



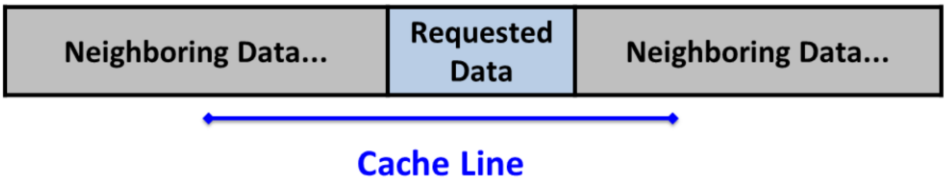
A Quick Explanation of the Cache



- When there's a cache miss, the cache loads in the requested data as well as the data immediately next to it in memory – this area is called a ***cache line***

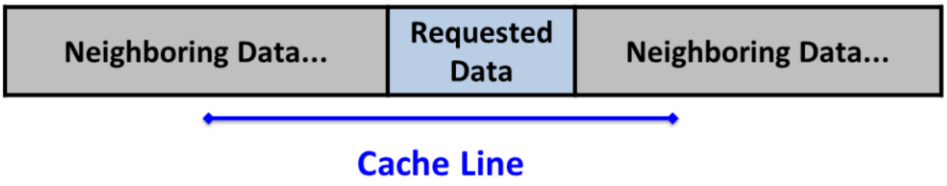
This is to optimize cache usage, with the idea that data which is being accessed is near other data that will also be accessed

A Quick Explanation of the Cache



- This means that there is a high desirability to have data next to the data you will be accessing soon!

A Quick Explanation of the Cache



- Most modern CPUs have a surprisingly small cache line – 64 bytes are loaded at a time
- (This is mostly because the L1 cache is quite small)

Vector Memory Layout

- In a vector, everything is right next to each other in memory

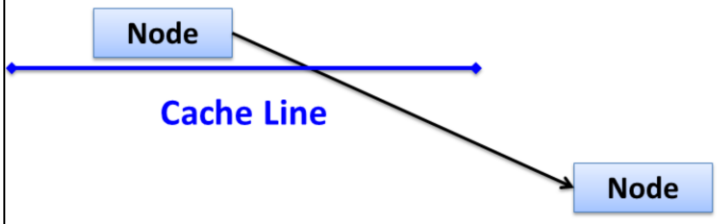
Index	0	1	2	3	4	5	6	7	8
Value	…	…	…	…	…	…	…	…	…

Cache Line

- Assuming ints…when index 0 is loaded into the cache, you get indices 1-15 for free
- If each cache miss is 200 cycles, you're amortizing it over 16 elements
- So with a vector 200/16 = **12.5 cycles/element**

Linked List Memory Layout

- By its nature, a linked list is all over the place in memory



- Thus one cache miss (200 cycles) will usually yield only 1 element ☹

Here's a Quote

"Modern CPUs employ pipelining as well as techniques like hardware threading, out-of-order execution and instruction-level parallelism to utilize resources as effectively as possible. In spite of this, some types of software patterns and algorithms still result in inefficiencies. **For example, linked data structures are commonly used in software, but cause indirect addressing that can defeat hardware prefetchers. In many cases, this behavior can create bubbles of idleness in the pipeline while data is retrieved and there are no other instructions to execute**" – Intel on "How to Tune Applications"

Disclaimer

- This ***does not mean*** that Big-O is useless
- Understanding algorithmic complexity is important, and for example an $O(n^3)$ algorithm is almost always going to perform worse than an $O(n)$ one

Disclaimer

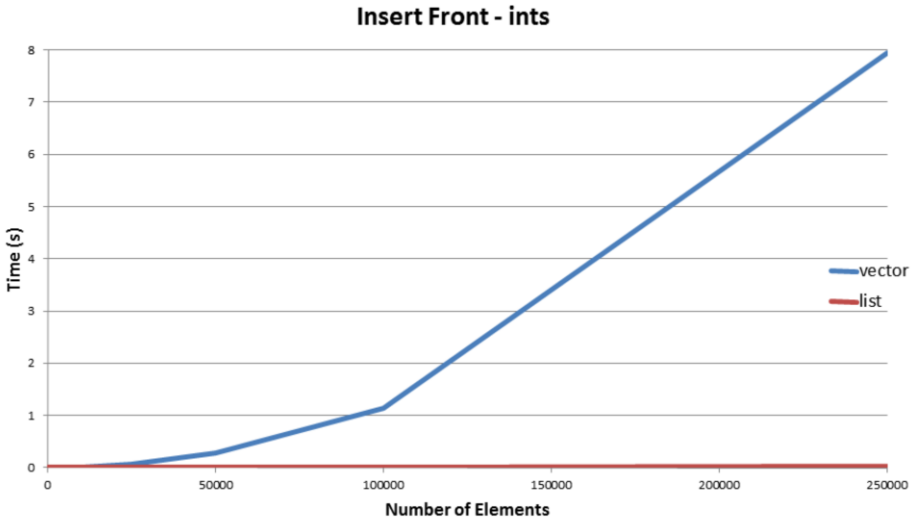
- ***However*** – Big-O is not the end-all be-all illustration of relative performance, especially when there are ties. Don't blindly accept Big-O as canon.

- When analyzing performance, it's perfectly reasonable to have a hypothesis based on Big-O, but be sure to test your hypothesis!

Let the Linked List Win – Insert to Front

- Insertion to the front is one metric where a linked list should win
- Vector is guaranteed to always have to copy ***N*** elements, whereas linked list is constant time
- Let's see what happens...

Insert Front – ints



Run-Length Encoding (RLE)

- Very simple form of compression
- If you have a string, repeated values are encoded as a "run" of that value.
- For example:

a	a	a	a	a	a	b	b	b	b

- Can be encoded as:

6	a	4	b

- This type of compression works well for data which repeats a lot (for instance, some image formats and MPEG compression uses RLE for some of its data)

USC Viterbi School of Engineering — University of Southern California

60% compression

Naïve RLE Problem

- With regular RLE, strings that have no runs are substantially longer
- This:

a	b	c	d	e

- "Compresses" to this:

1	a	1	b	1	c	1	d	1	e

Negative Runs

- One solution is to support "negative" runs. When a run is negative (-x), that means there are x values in a row which are unique
- So our previous example can be:

a	b	c	d	e

- Compressed to this:

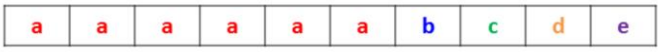
-5	a	b	c	d	e

USC Viterbi School of Engineering

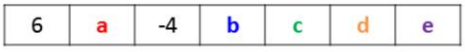
University of Southern California

It's still longer, but at least it's only one value longer.

Combined RLE Example



- Encodes to:



- Any questions??

60% compression