Administrative notes

• **Clickers**
  • Updated on Canvas as of people registered yesterday night.
  • REEF/iClicker mobile is not working for everyone. Use at your own risk.
  • If you are having trouble registering a physical clicker, look at the suggestions posted in the Announcements tab on Canvas. If you are still having trouble after that, let me know.

• In the News individual call #1 due January 17 (look at the course website for more details about what you need to do for that)

• Two reading quizzes due next Wednesday (the 17th)

• Reminder: optional course survey due January 17: [https://survey.ubc.ca/s/cpsc-100-student-experience-pre-2017w1/](https://survey.ubc.ca/s/cpsc-100-student-experience-pre-2017w1/)

• Exercises posted for hardware and algorithms
Administrative Notes

There is a general schedule of when pre-reading quizzes will happen but a lot of it depends on when we get through topics in class (sometimes we fall behind a bit).
A) Would you prefer that all the reading quizzes be released now (but not available for you to take) with the understanding that the dates may be moved around?

B) Would you prefer we release the quizzes as we go (typically you have about a week to complete the quiz). Those quiz due dates are certain and (99% of the time) won’t be moved.
Administrative Notes

The midterms are currently each worth 10% (we have 2 midterms in total). Feedback from the previous semester indicated that students wanted the midterms to be worth more.

Would you like to have each midterm be worth 15% and the final be worth 30%?
NOTE: You still have to pass both the project and the final to pass the course.
A) Yes
B) No
How many memory slots are needed for cards for Simple Sort for 7 cards? (don’t forget swapping!)

A. 7
B. 8
C. 14
D. 15
E. Other

When counting, ignore things like markers and dividers

Simple Sort Reminder:
1. Place the unsorted cards in the top row
2. Repeat steps 3 through 6 until no unsorted card remains
   3. Initially mark the first (leftmost) card
   4. Working right from the second card, if the current card is smaller than the marked card, move the marker to the current card
   5. Move the marked card to the sorted hand
   6. Put a “Max” (upside down) card in the empty unsorted slot
3. Stop
How many memory slots are needed for cards for Selection Sort for 7 cards (don’t forget swapping!)

A. 7
B. 8
C. 14
D. 15
E. Other

When counting, ignore things like markers and dividers

B. 8

Selection sort reminder:

1. Deal the unsorted cards
2. Put a divider at the left of the unsorted cards
3. Repeat steps 4 through 7 until one unsorted card remains
   4. Initially mark the first (leftmost) card
   5. For each card to the right of the second card:
       If the card is smaller than the marked card, move the marker to the current card
   6. Swap the marked card with the first unsorted card (just to the right of the divider)
   7. Advance the divider to the right one card
4. Stop
How many memory slots are needed for cards for **Insertion Sort** for 7 cards (don’t forget swapping!)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A. 7</td>
<td>When counting, ignore things like markers and dividers</td>
</tr>
<tr>
<td>B. 8</td>
<td>Insertion sort reminder</td>
</tr>
<tr>
<td>C. 14</td>
<td>1. Deal the unsorted cards</td>
</tr>
<tr>
<td>D. 15</td>
<td>2. Put a divider after the first card</td>
</tr>
<tr>
<td>E. Other</td>
<td>3. Repeat steps 4 through 6 until no card is to the right of the divider</td>
</tr>
<tr>
<td></td>
<td>4. Select the first card to the right of the divider</td>
</tr>
<tr>
<td></td>
<td>5. Swap this card to the left until it arrives at the correct sorted position</td>
</tr>
<tr>
<td></td>
<td>6. Advance the divider to the right one card</td>
</tr>
<tr>
<td></td>
<td>7. Stop</td>
</tr>
</tbody>
</table>
Space wrap up

Simple sort performed roughly twice as badly as either Selection Sort or Insertion Sort.
How about time?

- There are lots of things that we can look at that take time
  - Copies
  - Swaps
  - Comparisons
- Let’s concentrate on comparisons
How many comparisons are needed for Simple Sort (With 7 cards)

A. 7  
B. 8  
C. 21  
D. 42  
E. Other

Simple Sort Reminder:
1. Place the unsorted cards in the top row
2. Repeat steps 3 through 6 until no unsorted card remains
3. Initially mark the first (leftmost) card
4. Working right from the second card, if the current card is smaller than the marked card, move the marker to the current card
5. Move the marked card to the sorted hand
6. Put a “Max” (upside down) card in the empty unsorted slot
7. Stop
How many comparisons are needed for Selection Sort (With 7 cards)

A. 7
B. 8
C. 21
D. 42
E. Other

Selection sort reminder:
1. Deal the unsorted cards
2. Put a divider at the left of the unsorted cards
3. Repeat steps 4 through 7 until one unsorted card remains
4. Initially mark the first (leftmost) card
5. For each card to the right of the second card:
   - If the card is smaller than the marked card, move the marker to the current card
6. Swap the marked card with the first unsorted card (just to the right of the divider)
7. Advance the divider to the right one card
8. Stop
How many comparisons are needed for Insertion Sort (with 7 cards)?

A. < 10

B. 10 to 14

C. 15 to 19

D. 20 to 24

E. More than 24

The answer is: it depends!

When counting, ignore things like markers and dividers.

Insertion sort reminder:
1. Deal the unsorted cards
2. Put a divider after the first card
3. Repeat steps 4 through 6 until no card is to the right of the divider
4. Select the first card to the right of the divider
5. Swap this card to the left until it arrives at the correct sorted position
6. Advance the divider to the right one card
7. Stop

Computational Thinking
http://www.ugrad.cs.ubc.ca/~cs100
How many comparisons are needed for Insertion Sort? Group discussion.

• Which initial ordering of the cards would result in the maximum number of comparisons?
  When the cards are ordered in the opposite order that you want them in. (E.g., 10, 9, 8, 7, 6 when you want them in order from smallest to largest.

• Which initial ordering of the cards would result in the minimum number of comparisons?
  When the cards are ordered in the same order that you want them in. (E.g., 6, 7, 8, 9, 10 when you want them in order from smallest to largest.
On average, Selection Sort and Insertion sort use roughly half as many comparisons as Simple Sort.
Quick Sort and Merge Sort are even faster! Their (purple) plots lie close to the x axis.

![Graph showing comparisons for sorting algorithms](https://en.wikipedia.org/wiki/Sorting_algorithm)

- **# comparisons (in millions)**
- **# items to sort**

- Simple Sort
- Selection & Insertion Sorts
- Quick & Merge Sorts

[Read more](https://en.wikipedia.org/wiki/Sorting_algorithm)
That’s a big difference!

• On average, Selection Sort and Insertion sort have roughly half as many comparisons and use roughly half as much space as Simple Sort.

• Think about what that looks like when you have a lot of cards

![Graph showing number of comparisons per number of cards for Simple Sort and Selection & Insertion Sorts](chart.png)
Learning Goals

• CT Building Block: Students will be able to define the difference between hardware, applications, and the operating system.

• CT Building Block: Students will be able to give examples of a problem for which there are different algorithms, give examples of cases where one algorithm works better than the other, and reason about which algorithm is likely to work better overall.

• CT Building Block: Students will be able to evaluate and compare algorithms in terms of its efficiency (time and space requirements).
From algorithms to code: How do programs work?
Learning Goals

• [CT Building Block] Explain what a variable is in computer programming.

• [CT Building Block] Be able to trace through code using sequences of instructions, variables, loops, and conditional statements in short programs specified in a visual programming language such as Snap, or in other clearly expressed processes (which may or may not be computer related)

Note: Look carefully: it says be able to trace code, not write code. In order to help you do this, you will write a small amount of code in lab. You will not, however, be asked to write code on exam.
This is \textit{not} a programming course.

- That being said, we do want you to understand a little about how programs work.
- We’ll cover a small amount of basic concepts in class and you’ll work on a visual language in lab – Snap!
  \begin{itemize}
    \item \texttt{http://snap.berkeley.edu/snapsource/snap.html}
  \end{itemize}
At the highest level

• Programs are a way of encoding algorithms in a precise enough way for computers to understand the instructions
Programmers use a high level language like Snap, Scratch, Python, C++, Java, Racket.

These languages may look very different.
Another program called a *compiler* or an *interpreter* takes a high level language and translates it into something that looks about the same, regardless of which high level language is used.

```assembly
.CODE
.demomain:
    REPEAT 20
    switch rv(nrando, 9) ; generate a number between 0 and 8
    mov ecx, 7
    case 0
        print "case 0"
    case ecx ; in contrast to most other programming languages,
        print "case 7" ; the Masm32 switch allows "variable cases"
    case 1 .. 3
        .if eax==1
            print "case 1"
        .elseif eax==2
            print "case 2"
        .else
            print "cases 1 to 3: other"
        .endif
    ...
```
Another program called an assembler translates this into machine language.

Low-level machine languages are at the level of the computer’s hardware:

- `load 3, 7000` Load data #7000 from cache into register 3
- `load 4, 7001` Load data #7004 from cache into register 4
- `add 3, 4, 3` Add registers 3 and 4 and put in register 3
- `store 3, 7002` Store the data from register 3 in slot #7002
- `jmp 20` Go ahead in the instructions by 20
But many of the concepts are the same

- We’ll cover just enough so that you have an idea of how programs work
- If you want to learn more, you can take a class like CPSC 103 or CPSC 110
- For now, we’ll just stick with some examples like…
Blackjack!

• We’re going to model what needs to be done to play blackjack
• To do that, we need to make sure that everyone understands how blackjack works.
• In a minute, we’ll play a couple of rounds of blackjack in groups to make sure that everyone knows the rules.
  • There are more complex rules – we’re keeping it very simple
Blackjack as we know it
(simplified from the real world)

Ground rules:

• In blackjack, there is a dealer and one player.
• The player places a bet to get into the game.
• Face cards are worth 10. Aces can be worth 1 or 11.
• The goal is to get as close to 21 points as possible.
• Anything over 21 points loses.
• The dealer wins all ties.

Play:

• The dealer and the player each start with 2 cards; the dealer has 1 face down.
• The player is allowed to hit (ask for more cards) as many times as long as the total is under 22.
• The player then stands.
• The dealer will then take as many hits as necessary to get to more than 16 points.
Great! Now we’re ready to design a small Blackjack application

• Let’s take a look at one that some TAs and I put together.
• Note: we’re not going to look at ALL the code, just enough to get the idea of some key concepts.
There’s a lot to keep track of!

- Let’s start by listing all of the things that we need to keep track of, e.g., how many points the player’s hand is worth
- In a group, discuss some things we need to keep track of
  - Sum of the player’s cards
    - Stop the game if it goes over 21
  - \( J = 10 \)
    - \( Q = 10 \)
    - \( K = 10 \)
  - Player’s amount of money left
  - Bet placed per round