Linux Commands][Counting Primitive Operations][Time Requirements for Algorithms][Misc][Highest Value in an Array][Linear Search][Binary Search][Bubble Sort][Selection Sort][Pointers][Structures][Strings-Requires <string> header] String/Numeric Conversion Functions – Requires cstdlib header file - atoi – converts c-string to an int value, returns the value, atoll converts to a long value, returns the value, atof – converts to a double value, returns the value, itoa – converts 1st int parameter to a c-string stores it in 2nd parameter. "Source code > Preprocessor > Modified Source Code > Compiler > Object Code > Linker > Executable Code" "fstream for file access, ifstream for input from a file, ofstream for output to a file, fstream for input from or output to a file. Define file stream objects – ifstream infile; ofstream outfile;, infile.open ("inventory.dat"); outfile.open ("reports.txt"); << to send data to a file, >> to copy data from file to variables. Make sure to close file – infile.close(), outfile.close();" Passing by Reference – Changes to a reference variable are made to the variable it refers to, use reference variables to implement passing parameters by reference."

```
Struct Student
{
   int studentID;
   string name;
   short yearInSchool;
   double gpa;
};
Defining Variable – use structure tag as type.
Student bill;
struct variable can be initialized when defined:
Student s = {11465, "John", 2, 3.75);
Arrays of Structures – const in NUM_STUDENTS = 20;
Student stuList [NUM_STUDENTS];
cout << stuList[5].studentID;
Nested Structures –
Struct PersonInfo
{   string name, address, city;
};
Struct Student
{   int studentID; PersonInfo pData; short yearInSchool;
Double gpa;
};
Student s; s.pData.name = "Joanne"; s.pData.city = "Tulsa";
Can use structure pointer operator to eliminate ()
Cout << stuPtr ->studentID; instead of (*stuPtr)
```

Each variable in program is stored at a unique address, use address operator &, to get "Each variable in program is stored at a unique address, operator &, to get address of a variable: : int num = -99; cout << # //prints address in hexadecimal. int *intptr; read as "intptr can hold the address of an int" Spacing in definition does not matter: int * intptr; int* ptr;. Assigning an address to a pointer variable: int *intptr; intptr = #

Array elements can be accessed in many ways: array name and [] – vals[2] = 17; pointer to array and [] – valptr[2] = 17; array name and subscript arithmetic - *(vals + 2) = 17; pointer to array and subscript arithmetic - *(valptr + 2) = 17; Initializing Pointers – int num, *numptr = # int vals[3], *valptr = val; Pointers as Function Parameters – void getNum(int *ptr); //ptr is pointer to an int. Dynamic Memory Allocation – Can allocate storage for a variable while program is running, uses new operator to allocate memory: double *dptr; dptr = new double; new returns address of memory location. Use delete to free dynamic memory: delete fptr; use [] to free dynamic array: delete [] arrayptr;

Total 7n – 1 (Worst case)

```
Algorithm arrayMax (A,n)
currentMax ←A[0]          2
for I ←to n-1 do          2 + n
if A[i] > currentMax then 2(n-1)
currentMax ←A[i]          2(n-1)
increment count i}        2(n-1)
return currentMax         1
```

```
int count, highest;
highest = numbers[0]
for (count = 1; count < SIZE;
count++)
{    If (numbers [count] > highest)
     highest = numbers [count];
}
```

Selection Sort – Locate smallest element in array. Exchange it with element in position 0. Locate next smallest element in array, exchange it with element in position 1. Continue until all elements are arranged in order.

```
void selectionSort (int array[], int size)
{
  int startScan, minIndex, minValue;

  for (startScan = 0; startScan < (size – 1);
startScan ++)
  {
    minIndex = startScan;
    minValue = array[startScan];
    for (int index = startScan + 1; index < size;
index ++)
    {
      If (array[index] < minValue)
      {
        minValue = array[index];
         minIndex = index;
      }
    }
    array[minIndex] = array[startScan];
    array[startScan] minValue;
    }
}
```

Binary Search, requires elements to be in order, divides the array into three sections – middle element, elements on one side of the middle element, and elements on the other side of the middle element.

```
int binarySearch(int array[], int numElems, int
value)
{
  int = first = 0, last = numElemes – 1, middle,
position = -1, bool found = false;

while (!found && first <= last)
{
  middle = (first + last) / 2;
  position = middle;
}
else if (array[middle] > value) // lower half
  last = middle – 1;
else
  first = middle + 1; // upper half
}
return position;
}
```

cctype header file, isalpha – true if Is a letter, false otherwise, isalnum – true if Is a letter or digit, false otherwise, isdigit true if Is a digit 0-9, false otherwise, islower – true if Is lowercase letter, false otherwise, isprint – true if Is a printable character, false otherwise, ispunct true if Is a punctuation character, false otherwise, isupper true if Is an uppercase letter, false otherwise, isspace – true if Is a whitespace character, false otherwise. "if (isalpha(input)), char ch1 ='h'; cout << toupper(ch1); displays 'H'

Bubble Sort - Compare 1st two elements, if out of order, exchange them to put in order, move down one element, compare 2nd and 3rd elements, exchange if necessary. Continue until end of array.

```
void soryArray(int array[], int size)
{
  bool swap;
  int temp;

  do
  {
    swap = false;
    for (int count = 0; count < size – 1); count
++)
{
  If (array[count] > array[count + 1])
  {
    temp = array[count];
    array[count] = array[count + 1];
    array[count + 1] = temp;
    swap = true;
  }
}
} while (swap);
}
```

Linear Search aka Sequential search. Starting at the first element, examining each element until it locates the value it is searching for.

```
int searchList(int list[], int numElems, int value)
{
Int index = 0; int position = -1; bool found =
false;
while (index < numElems && !found)
{
If (list[index] == value)
{
found = true; position = index;
}
Index++;
}
return position;
}
```

"Sequential search – 0(n), constant multiple of n" "Binary search – in the average case and the worst case, $0(\log_2 n)$" "Selection sort, $O(n^2)$" "Bubble Sort, Best O(n) – Worst – $O(n^2)$" "Sequential search, Worst case O(n)"

"Ls = list" "mkdir = make directory" "cd = change directory" ".= current directory" "..= parent directory" "pwd = print working directory" "cp = copy" "mv = move" "rm = remove" "rm = remove directory" "clear= clear screen" "cat = concatenate" "less = writes only one page" "head = writes first 10 lines" "tail = write last 10 lines" "grep = word search" "we = word counter" "cat > File = writing to file" "cat >> File = adding to file" "*= wildcard : just one" "man = manuals" "aprops = approximate name of command" "chmod = change name of command" "chmod = change file mode" "kill = terminate process" "ps = process status" "bg = background suspended job" "fg = foreground suspended job" "command & = run command in background" "quota = check current quota" "df = reports space left" "du = out kilobytes of subdirectories" "gzip = reduces size of files" "zeat = read gzipped file" "file = classifies files according to data type" "diff compares content of two files" "find = find file using given words" "history = shows command history" "cat list1 list2 > biglist" " to remove read, write, and execute permissions - chmod go-rwx big list" " to give read and write permissions to all – chmod a+rw biglist" "Building the package – make, make check, make install