DS 5010 Assignment 4

Saurabh Yelne

10/7/2018

1. Write a function that accepts a vector of integers, and returns a new sorted vector using selection sort.

```
selection_sort = function(v) {

  for(i in 1:length(v)) {
    min = i

    for(j in i:length(v)) {
      if(v[j] < v[min]) {
        min = j
      }
    }
    temp = v[i]
    v[i] = v[min]
    v[min] = temp
  }
  return(v)
}

v <- c(100,2,4,55,67,83,12,34,45)


selection_sort(v)

## [1]   2   4  12  34  45  55  67  83 100
```

2. Write a function that accepts a vector of integers, and returns a new sorted vector using insertion sort

```
insertion_sort <- function(x)
{
   for(i in 2:(length(x)))
   {
      value <- x[i]
      j <- i - 1
      while(j >= 1 && x[j] > value)
      {
         x[j+1] <- x[j]
         j <- j-1
      }
      x[j+1] <- value
   }
```

```
  return(x)
}
v <- c(100,2,4,55,67,83,12,34,45)
insertion_sort(v)

## [1]   2   4  12  34  45  55  67  83 100
```

3. Write a function that accepts a vector of integers, and returns a new sorted vector using bubble sort

```
bubble_sort <- function(x){
  n<-length(x)
  for(j in 1:(n-1)){
    for(i in 1:(n-j)){
      if(x[i]>x[i+1]){
        temp<-x[i]
        x[i]<-x[i+1]
        x[i+1]<-temp
      }
    }
  }
  return(x)
}
v <- c(100,2,4,55,67,83,12,34,45,35)
bubble_sort(v)

##  [1]   2   4  12  34  35  45  55  67  83 100
```

4. Write a function that accepts a sorted vector and searches for a specific number in the list. Make sure you algorithm runs in log2(n) time and not linear time of O(n)? What is the name of this search? Prove with a small example that the runtime is actually O (log2n).

```
binary_search <- function(v, value){
  result <- list(counter=0,val=0)
  lower <- 1
  upper <- length(v)
  while(lower<=upper){
    result[[1]][1] <- result[[1]][1]+1
    middle <- round((upper+lower)/2)
    if (v[middle]==value){
      result[[2]][1] <- result[[2]][1]+middle
      return(result)
    }else if(v[middle]<value){
      lower <- middle+1
    }else {
      upper <- middle-1
    }
  }
}
```

```r
h <- c(1:128)
binary_search(h,value = 99)
```

```
## $counter
## [1] 7
##
## $val
## [1] 99
```

As we can see if input is 128 the counter is 7. So the function's runtime is log(n).

5. Write a function that would accepted an integer argument called size, and would return a vector of that size populated with values of NA.

```r
vector_NA <- function(size= int){
  vec <- rep(NA,times=size)
  return(vec)
}
vector_NA(size = 10)
```

```
##  [1] NA NA NA NA NA NA NA NA NA NA
```

6. Write a function called pop that would return the last element in that vector that is not NA and would replace the element with NA in the vector.

```r
pop <- function(vec){
  result <- list()
  if(length(vec)==0){
    return(NA)
  }else if (is.na(vec[1])){
    return(NA)
  }else{
    for(i in 1:length(vec)){
      if(i == length(vec)&&is.na(vec[i])){
        result$pop_element <- vec[i-1]
        vec[i-1] <- NA
        result$vector_after_pop <- vec
        return(result)
      }else if(i==length(vec) && !is.na(vec[i])){
        result$pop_element <- vec[i]
        vec[i-1] <- NA
        result$vector_after_pop <- vec
        return(result)
      }
    }
  }
}

v <- c(1,3,5,NA)
a<- pop(v)
a
```

```
## $pop_element
## [1] 5
##
## $vector_after_pop
## [1]  1  3 NA NA
```

7. Write a function called push that accepts an element and replaces the first NA element in the vector with the value that is passed, (essentially adding to the end of the vector). You cannot use the concatenate function. push2 <- function(v,element){ if (length(v) == 0){ return v; } for (i in 1:(length(v))){ if (is.na(v[i])){ v[i] = element; return v(v); } }

return v; }

```
push <- function(v,element){
  for(i in seq_along(v)){
    if(is.na(v[i])){
      v[i] <- element
      return(v)
    }else if (i== length(v) & !is.na(v[i])){
      print("Error, vector has no space to push")
    }
  }
}
```

8. Using your functions from above write a function that determines whether every opened bracket is matched with a corresponding closed bracket. The run time of this algorithm has to be O(n). Your function should accept a string.

```
pop1 <- function(vec){

  if(length(vec)==0){
    return(NA)
  }else if (is.na(vec[1])){
    return(NA)
  }else{
    for(i in 1:length(vec)){
      if(is.na(vec[i])){
        temp <- vec[i-1]
        vec <-  (vec[-(i-1)] )
        return(vec)
      }else if(i==length(vec) && !is.na(vec[i])){
        temp <- vec[i]
        vec <-  (vec[-i])
        return(vec)
      }
    }
  }
}
v <- c(1,4,66)
pop1(v)
```

```
## [1] 1 4
```

```
balance_brac <- function(char){
  input <- strsplit(char,"")
  d <- rep(NA, length(input[[1]]))
  for(i in 1:length(input[[1]])){
    if(input[[1]][i] == "("){
      d <- push(d,"(")
    }else if (input[[1]][i] == ")"){
      if(is.na(d[1])){
        return(FALSE)
      }else{
      d <- pop1 (d)
    }
  }
  }
  if(is.na(d[1])){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

-For example this should return true:

"((2 + 2) * 3)"

```
balance_brac("( ( 2 + 2) * 3 )")

## [1] TRUE
```

-This should return false

"((2 + 2) * 3" -> missing closing bracket: -> This should return false:

```
balance_brac("( ( 2 + 2) * 3")

## [1] FALSE
```

"(2+2) * 3)" -> missing opening brakcet

-> This should return false:

```
balance_brac("(2+2) * 3 )")

## [1] FALSE
```

"2 + (2 (+ 3" -> missing two closing brackets.

```
balance_brac("2 + ( 2 ( + 3")

## [1] FALSE
```

Other examples that would return false:

"((( ))"

```
balance_brac("( ( ( ) ) ")

## [1] FALSE
```

"(2) (3) (2"

```
balance_brac("( 2 ) ( 3 ) ( 2")

## [1] FALSE
```

9. Can you think of a way to rewrite the push function but avoid writing a for loop that finds the first NA element in the vector?

```
push <- function(l,ele){
  if(is.na(l[[2]][l$index+1])){
    l[[2]][l$index+1] <- ele
    l$index <- l$index+1
  }else{
    print("Error, no space to push")
  }
  return(l)
}

b<-list(index=4,c(1,2,3,6,NA))
push(b,99)

## $index
## [1] 5
##
## [[2]]
## [1]  1  2  3  6 99
```