alds hade

## Ruby

- Developed by Yukihiro Matsumoto, mid-1990s
- Object-oriented
- Dynamically-typed / weakly-typed
- Can be used in many fields

## Ruby

- ✓ Like Python, it’s very easy to learn
- ✓ Shares similar characteristics
- ✓ It abstracts a lot of things from the programmer
- ✓ Would I recommend this for beginners? Yes

Two sorts implemented:

1. Merge sort
2. Quicksort

## Merge sort

1. Divide the unsorted list into smaller and smaller lists until each list only has 1 element
2. Repeatedly merge sublists to produce bigger lists until there is nothing to merge anymore

## Merge sort (divide)

```
if array.count <= 1
    return array
end

mid = array.count / 2
left_partition = mergesort array.slice(0, mid)
right_partition = mergesort array.slice(mid, array.count - mid)
```

# Merge sort (conquer)

```
array = []
left_lead = 0
right_lead = 0
while left_lead < left_partition.count && right_lead < right_partition.count
    a = left_partition[left_lead]
    b = right_partition[right_lead]

    if a <= b
        array << a
        left_lead += 1
    else
        array << b
        right_lead += 1
    end
end
```

# Merge sort (conquer)

```
while left_lead < left_partition.count
    array << left_partition[left_lead]
    left_lead += 1
end

while right_lead < right_partition.count
    array << right_partition[right_lead]
    right_lead += 1
end

return array
```

## Quicksort

1. Pick an element, call it the pivot
2. Create two sublists; all elements less than the pivot go to the left list, all elements greater than the pivot go to the right
3. Recursively apply the above steps to the left and right lists

## Quicksort (“main”)

```
if low < high
    p = partition array, low, high
    quicksort array, low, p-1
    quicksort array, p+1, high
end

return array
```

# Quicksort (partition)

```
i = low
j = high + 1
pivot = array[low]
while true
# Loop to increment i
    begin
        i += 1
        break if i==high
    end while array[i] < pivot
# Loop to increment j
    begin
        j -= 1
        break if j == low
    end while array[j] > pivot
    break if i >= j
    temp=array[i]
    array[i]=array[j]
    array[j]=temp
end

temp = array[low]
array[low] = array[j]
array[j] = temp
return j
```

```
Learning a New Programming Language: Ruby
by Alds Hade, ID# 11789190


[Merge sort]
[1] (random) = {4, 11, 12, 1, 10, 2, 5, 0, 3, -1, 7, 8, 6, 1, 9}
[1] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[2] (random) = {0, 10, 8, 5, 7, 2, 4, -1, 9, 1, 1, 11, 6, 12, 3}
[2] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[3] (random) = {7, 12, 8, 1, -1, 1, 0, 2, 10, 6, 9, 4, 11, 5, 3}
[3] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

```
Learning a New Programming Language: Ruby
by Alds Hade, ID# 11789190


[Merge sort]
[1] (random) = {4, 11, 12, 1, 10, 2, 5, 0, 3, -1, 7, 8, 6, 1, 9}
[1] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[2] (random) = {0, 10, 8, 5, 7, 2, 4, -1, 9, 1, 1, 11, 6, 12, 3}
[2] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[3] (random) = {7, 12, 8, 1, -1, 1, 0, 2, 10, 6, 9, 4, 11, 5, 3}
[3] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[Quick sort]
[1] (random) = {3, 4, 7, -1, 10, 9, 0, 2, 11, 5, 1, 8, 12, 6, 1}
[1] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[2] (random) = {9, 6, 7, 12, 0, 11, 8, 5, 1, 2, -1, 3, 10, 4, 1}
[2] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

[3] (random) = {3, 12, 10, -1, 1, 8, 7, 4, 1, 6, 11, 2, 0, 9, 5}
[3] (sorted) = {-1, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Evaluation criteria used

- ✓ Readability
- ✓ Writability
- ✓ Reliability
- ✓ Cost

Ruby, readability

✓ Ruby's dynamic typing,
lack of terminators, and
lack of braces
make it easy to read at a glance. It looks very clean.

✓ No need to separate the main function from the function declarations, so it can be disorganized; but most progress

x Dynamic typing can make it confusing to track what variables contain

Ruby, writability

✓ Very easy to write from scratch

✓ The notations are concise (e.g. for loops), so programs can be written quickly

? The main function is not explicitly declared

x ++ and -- are not present in Ruby

Ruby, reliability

✓ Good readability and writability

✓ Ruby is advertised as being “flexible” so
[1] programmers can solve problems such as sorting algos in different ways
[2] Ruby programs run until the error line

✓ Supports exception handling well

? There is type checking done when running variables, but no type checking done otherwise

Ruby, cost

✓ Free and easy to install

✓ Very easy to learn, but has unique constructs from Python or Java that require memorization in order to master Ruby

✓ No known cases of failure in professional use (education, entertainment, etc.)

x  Ruby is reported to be slower than other languages due to its weak typing

| Category | Ruby | MLG |
| --- | --- | --- |
| *Simplicity* | Simple but powerful | Hard to learn, not powerful |
| *Orthogonality* | Just enough | Too limited |
| *Data types* | Dynamic | Int / float / char / string / boolean |
| *Syntax design* | Clean, concise, clear | Poor |
| *Abstraction* | Supported | Supported |
| *Expressivity* | Good expressivity | No variations |
| *Type Checking* | Only when value is being used | Yes |
| *Error Handling* | Very flexible | Yes |
| *Aliasing* | No aliasing | Yes |